DOI:10.15774/PPKE.ITK.2023.008

GPU-related efficient visual information processing approaches

PhD dissertation

Anna Gelencsér-Horváth

Scientific advisors:
György Cserey, PhD
Pázmány Péter Catholic University
Kristóf Karacs, PhD
Pázmány Péter Catholic University

Pázmány Péter Catholic University
Faculty of Information Technology and Bionics
Roska Tamás Doctoral School of Sciences and Technology
Budapest, 2023

DOI:10.15774/PPKE.ITK.2023.008

DOI:10.15774/PPKE.ITK.2023.008

"Leave this world
a little better
than you found it."

Sir Robert Baden-Powell

DOI:10.15774/PPKE.ITK.2023.008

DOI:10.15774/PPKE.ITK.2023.008

Abstract

In my research I addressed the problem of efficiency in computer vision from two aspects. First, in terms of running time and efficient, parallel implementation. Second, I worked on the problem of reducing human effort concerning deep learning-based approaches for multi-animal segmentation and tracking.

The acceleration of computer vision algorithms by redesigning the steps that are identified as a bottleneck to take advantage of high-performance hardware such as Graphics Processing Units (GPUs) is an effective way to speed up visual information processing. I introduce a modified Cellular Particle Filter (CPF), which I mapped onto a GPU architecture. I developed this filter adaptation using a state-of-the-art CPF technique. Mapping this filter realization on a highly-parallel architecture led to a novel arrangement in the logical representation of the particles. In this process the original two-dimensional topology is reordered as a one-dimensional ring topology. I evaluated a proof-of-concept measurement on two widely used benchmark models with an NVIDIA Fermi architecture GPU. I compared the kernel and global running times of our approach to other state-of-the-art implementations, and to a CPU implementation as an additional reference on speed-up and error range of the serial (non-parallel) algorithm. Results demonstrate an effective and fast use of the particle filter in high-dimensional, real-time applications.

Automated annotation of many hours of surveillance videos can facilitate a large number of biological studies/experiments, which otherwise would not be feasible. Identity tracking and instance segmentation are crucial in several areas of biological research. Behavior analysis of individuals in groups of similar animals is a task that emerges frequently in agriculture or pharmaceutical studies, among others.

Although, solutions based on machine learning and deep learning generally perform well and thus are prevalent and preferred for image segmentation. However, in the case of tracking and instance segmentation of identical, unmarked instances (e.g., white rats or mice), even state-of-the-art approaches can frequently fail. To achieve high-quality

DOI:10.15774/PPKE.ITK.2023.008

segmentation of challenging frames, for example, for images with very similar object instances in occlusion, deep models need a significant number of training images that capture these types of features. However, manual annotation of these is both time-consuming and challenging due to the complex nature of the images. Therefore, I designed synthetic data generation techniques for the models used by the proposed method, which automatically generate and thus efficiently provide the required number of training images with challenging input features. I created a pipeline of deep generative models for identity tracking and instance segmentation of highly similar instances, which, in contrast to most region-based approaches, exploits edge information and consequently helps to resolve ambiguity in heavily occluded cases. I show that my approach greatly outperforms other state-of-the-art unsupervised methods in identity tracking and instance segmentation on unmarked rats in real-world laboratory video recordings.

DOI:10.15774/PPKE.ITK.2023.008

Összefoglalás

Kutatásomban a számítógépes látás hatékonyságával foglalkoztam két szempontból. Egyrészt a futási idő és a hatékony, párhuzamos megvalósítás kérdésével, valamint az emberi figyelem és időráfordítás csökkentésével a mélytanulás-alapú megközelítésekben több hasonló állat szegmentációja és követése során.

A vizuális információfeldolgozás felgyorsításának hatékony módja a gépi látás algoritmusainak gyorsítása a szűk keresztmetszetű lépések újratervezésével, hogy illeszkedjenek és kihasználják nagy teljesítményű hardverek, például a GPU architektúrák tulajdonságait. A state-of-the art celluláris particle filter (CPF) módszerből kiindulva elkészítettem és a GPU-ra leképeztem egy módosított particle filter algoritmust. Ahhoz, hogy a szűrő leképezhető legyen a nagymértékben párhuzamos architektúrára, a részecskék logikai reprezentációjának újraértelmezése volt szükséges. Ennek során az eredeti kétdimenziós topológiát átrendeztem egydimenziós gyűrűs topológiává. A *proof-of-concept* mérést két széles körben használt benchmark-modellen értékeltem ki egy NVIDIA Fermi architektúrájú GPU-n. Összehasonlítottam a megközelítésünk kernel- és globális futási idejét a legmodernebb implementációkkal, valamint egy CPU implementációval, amely megmutatja a nem párhuzamosított algoritmushoz képest elérhető sebességnövekedést a megőrzött hibatartomány mellett. Az eredmények igazolják a GPU CPF hatékony és gyors használatát nagydimenziós, valós idejű alkalmazásokban.

Sok órányi videó automatizált annotálása számos olyan biológiai vizsgálatot/kísérletet támogathat, amelyek máskülönben nem lennének megvalósíthatók. Az identitáskövetés és az állatok szegmentálása a felvételeken a biológiai kutatások számos területén kulcsfontosságú. Többek között a mezőgazdasági vagy gyógyszerészeti vizsgálatokban gyakran felmerülő feladat a hasonló állatok csoportjaiban lévő egyedek egyéni viselkedésének elemzése. Bár a gépi tanuláson és a mélytanuláson alapuló megoldások a szegmentálásban elterjedtek és kedveltek, mivel általában jól teljesítenek, azonos kinézetű, megkülönböztető jelöléssel nem ellátott

DOI:10.15774/PPKE.ITK.2023.008

példányok (pl. fehér patkányok vagy egerek) szegmentálásában és követésében még a legkorszerűbb megközelítések is gyakran kudarcot vallhatnak. A modellek tanításához, hogy képesek legyenek a kihívást jelentő képkockák jó minőségű szegmentációjára, például olyan képek esetén, ahol a nagyon hasonló állatok átfedő pozícióban vannak, jelentős számú olyan tanítóképre van szükség, amelyek a kihívást jelentő jellemzőket tartalmazzák. Ezek kézi annotálása azonban épp a képek összetett jellege miatt aprólékos és időigényes feladat. Ezért a módszerem által használt modellekhez olyan szintetikus adatgenerálási technikákat terveztem, amelyek automatikusan generálják és így hatékonyan biztosítják a szükséges számú, kihívást jelentő bemeneti jellemzőkkel rendelkező tanító képet. Létrehoztam egy mély generatív modellekből álló módszert, amely a régióalapú megközelítés mellett kihasználja az élinformációt, így az erősen átfedő esetekben is külön-külön szegmentálja a példányokat. Megmutatom, hogy az általam alkalmazott megközelítés jelentősen felülmúlja a többi korszerű, felügyelet nélküli módszert a valós laboratóriumi videófelvételeken szereplő, jelöletlen patkányok identitáskövetésében és példányszegmentálásában.

DOI:10.15774/PPKE.ITK.2023.008

Contents

DOI:10.15774/PPKE.ITK.2023.008

DOI:10.15774/PPKE.ITK.2023.008

List of Figures

DOI:10.15774/PPKE.ITK.2023.008

DOI:10.15774/PPKE.ITK.2023.008

List of Tables

DOI:10.15774/PPKE.ITK.2023.008

DOI:10.15774/PPKE.ITK.2023.008

List of Abbreviations

aGT	.....	Augmented Ground Truth
aRGB	.....	Augmented RGB
BCE	.....	Binary Cross Entropy
BG/FG	.....	Background / Foreground
BIPED	.....	Barcelona Images for Perceptual Edge Detection
BOT model	.....	Bearings-Only Tracking model
CPF	.....	Cellular Particle Filter
CPU	.....	Central Processing Unit
CUDA	.....	Compute Unified Device Architecture
GPU	.....	Graphics Processing Unit
GAN	.....	Generative Adverserial Network
HED	.....	Holistically-nested Edge Detection
HMM	.....	Hidden Markov Model
ID	.....	Identity
IID	.....	Independent and Identically Distributed
IoU	.....	Intersecion over Union
MCM	.....	Monte Carlo Methods
MIS	.....	Maximizing Importance Selection
MSE	.....	Mean Square Error
MT	.....	Mersenne Twister random generator
OF	.....	Optical Flow
PF	.....	Particle Filter
RSS	.....	Root Sum of Squares
SMCM	.....	Sequential Monte Carlo Methods
SIR	.....	Sequential Importance Resampling
SR	.....	Systematic Resampling

DOI:10.15774/PPKE.ITK.2023.008

DOI:10.15774/PPKE.ITK.2023.008

Chapter 1

Introduction

Since visual perception provides us with a significant amount of information about our surroundings, processing visual information to mimic human vision with computers is a widely researched area with many motivations. It is hard to think about a field where automatic visual information processing would not be useful, but typical areas such as security surveillance [1, 2], industrial quality assurance [3, 4], medical diagnostics [5–9], agriculture (veterinary) [10, 11], behavioral ecology [12], self-driving vehicle navigation [13–15] can highly exploit it. In computer vision, object detection, segmentation, and tracking segmented objects are essential in moving towards automated information processing, practically regardless of the context domain.

The speed of computer vision algorithms is critical in many applications including those that require real-time information processing, such as a navigation system. Although deep learning is now widespread, an end-to-end solution may require combining deep networks with traditional methods, depending on the task. Particle filters can be traced back to at least the 1990s [16] and they correspond to a robust approach to deal with nonlinear state-space models subject to additive noise, not restricted to Gaussian noise [17]. In computer vision, it is usually used to approximate time series of data derived from pixel information, such as object tracking, or directly on the visual input as high-dimension data [1, 7, 9, 18]. Even if each state only depends on the previous state (i.e. the sequence follows Markovian dynamics[19]) a Kalman filter [20] is suboptimal for state estimation due to the nonlinearity of the state dynamics and non-Gaussian noise [21, 22]. Furthermore, an analytic solution is often not available. In contrast, Sequential Monte Carlo Methods (SMCM) offer a probabilistic framework

DOI:10.15774/PPKE.ITK.2023.008

that is suited to non-linear and non-Gaussian state-space models. Particle filters (PF) are both part of the SMCM algorithm family and can be considered an extension of the Kalman filter. The potential use of particle filtering goes far beyond predicting time series, such as in financial mathematics [23, 24] or position tracking, despite the challenge of applying particle filtering to high-dimensional systems [25]. It is used in many approaches where the input is an image, or a series of images, including image reconstruction [26], object detection [27], navigation [28], segmentation [9, 29], contour detection [7], and in tracking with occlusion [30, 31].

A benchmark time series and its observation [17] is illustrated in Figure 1.1.

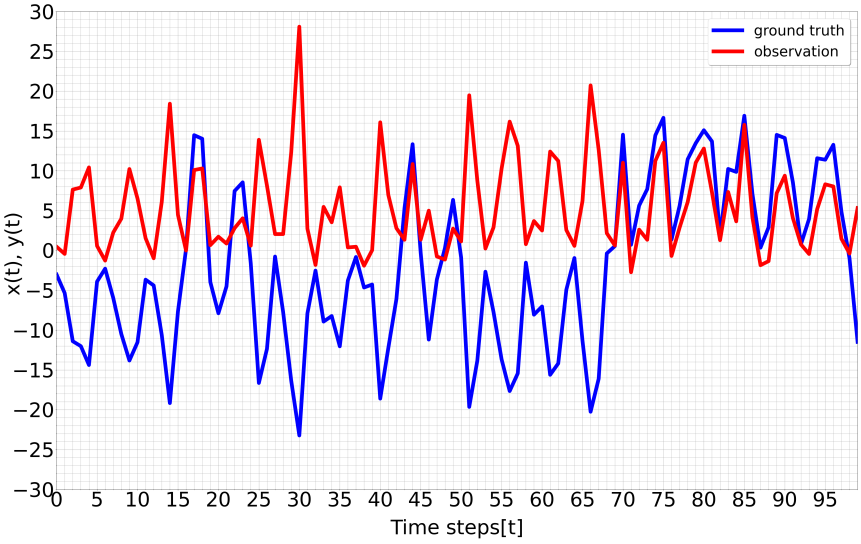

Figure 1.1: A time series (ground truth with blue) and its observation (red) of a particle filter benchmark model [17].

The PF algorithm is computationally expensive due to the resampling step according to the *complete* cumulative distribution [32]. The efficient and high prediction quality implementation to parallel architectures has been widely researched [33–42]. Graphics processing units (GPUs) represent an attractive implementation platform as they have high computational efficiency, while the price of a device is relatively low. Since GPUs have become widespread thanks to the video game industry and developed rapidly, the computational capacity increased by leaps and bounds. This increased capacity can

DOI:10.15774/PPKE.ITK.2023.008

only be well exploited if algorithms are adapted to the characteristics of the hardware architecture being used. Nevertheless, the method used for resampling has a high impact on the prediction quality. As a comparison shows in [43], different resampling strategies offer trade-offs between speed and prediction quality. Therefore, an adequate parallel implementation of PF that retains local connections and the information exchange among the particles during resampling can achieve a remarkable speed-up with no degradation of the estimation quality. During my research, I addressed the acceleration of the particle filtering algorithm, to re-design it to fit GPU architectures of the time the most efficiently [A1, A3].

My research goal was to introduce a fast particle filter with sequential importance resampling (SIR) [44] and provide a novel method that allows the implementation on GPUs to retain high-quality prediction. The classical resampling algorithm of this process needs N processors to reduce its computational need from $O(N)$ to $O(log(N))$, as long as the complete cumulative distribution is required for any particle, so this particle filter was considered unsuitable for parallelism. We can consider resampling as a key point in SIR particle filtering. Previous literature has presented approaches to parallelization with a speed-quality trade-off, so a need arose for a novel approach that could overcome the information loss issue. Cellular particle filter(CPF) [45] introduces a promising approach for the resampling problem by changing the logic representation of the PF to a two-dimensional (2D), locally connected grid inspired by the cellular neural network (CNN) architecture [46]. Each element in the grid is connected to all of its eight neighbors enabling rapid local information flow. The critical resampling step can then be performed on a subset in a radius r neighborhood. Due to the CNN-type representation and the decreased dimension of resampling sets, CPF offers a solution to the problem of reduced local information change, which is stated in [34]. Although the prediction is of the same quality as for sequential implementation, this representation is not optimal in terms of exploiting GPU architecture to achieve high efficiency. Hence, I intended to re-design the algorithm to achieve an efficient implementation that fits the characteristics of the GPUs and thus, can exploit and computational capability and speed of GPUs.

DOI:10.15774/PPKE.ITK.2023.008

In addition to the importance of increasing the efficiency of the core algorithms, what really matters in engineering tasks is the total time required for executing a process, including data acquisition and preparation. In addition to optimizing for the speed of individual components, such as a particle filter used for tracking, we also need to consider the need for suitable input usually provided by segmentation. Forming regions by organizing and labeling each pixel in the image into groups that belong to the same object, segmentation is a strategy for understanding and processing visual information. Therefore, I focused on segmentation and contour detection, as it is challenging for region-based methods based on pixel similarity to avoid merging similar adjacent objects or parts thereof. In addition to considering region similarities based on colors on an over-segmented image, created with a segmentation based on Mean Shift [47], I intended to exploit contour detection based on basic geometrical properties of the segments [A4, A5].

However, in addition to image processing and machine learning methods, over the past decade, there has been a rapid development of deep learning-based approaches for segmentation [48]. After a break between 2015 and 2021, I began investigating the segmentation of neighboring, highly similar areas using deep neural networks. The research was motivated by surveillance videos of unmarked rats, where even state-of-the-art algorithms failed in multi-animal tracking without a single ID switch due to heavy occlusions. Deep networks are effective tools for retrieving visual information automatically, and if sufficient training data is available, the final model can learn a high-dimensional representation of important image features - similar to the human brain. Compared to traditional algorithms, deep networks require a task-dependent amount of labeled data for training. Therefore, we can consider the quality of prediction and the time spent on data acquisition and annotation as a trade-off. For complex scenes, the collection of training data may be beyond the possibilities in terms of computational or human resources. Even if the required amount of data of adequate type is available, annotation still needs considerable time. Therefore, in terms of efficiency, we need to consider not only the quality and speed of the prediction, but also the time required to

DOI:10.15774/PPKE.ITK.2023.008

collect, prepare, and annotate the data for training. In the case of so-called end-to-end networks [49, 50], the training of a single, even complex structure provides the final prediction for the raw, not pre-processed input data. However, defining such a network can be a significant challenge, as it depends on the specific task and the characteristics of the input images. Composite AI [51, 52] is an approach that can help overcome limitations of end-to-end deep networks. For a complex task, deep learning, machine learning, image processing, higher-level logic, and even domain-specific knowledge can be used to create a pipeline that ensures a high-quality performance for challenging scenes with multiple unknown objects, occlusions, noises, and similarities, among others. Training the network on synthetic data, where the ground truth labeling is available by construction, saves considerable time and effort. However, the distribution and image characteristics of the synthetic training data must be sufficiently close according to an appropriate metric to those of the candidate inputs to ensure high-quality prediction and to avoid overfitting to artifacts, which is often highly challenging.

Behavior analysis of individuals in groups of similar animals is a task that emerges frequently in agriculture, behavioral ecology, or pharmaceutical studies, among others. Automated annotation of many hours of surveillance videos can facilitate a large number of biological studies, which otherwise would not be feasible. In medical and pharmaceutical research, visual tracking of treated animals is used for behavior analysis to detect the effects of drugs or other treatments. An example of such research is the investigation of the effects of pharmacological treatment on autism, specifically examining how the drugs improve social behavior, reduce rejection of approach by other instances, or decrease aggressive reactions, and whether it affects exploratory skills. Visual observation is an easily accessible method compared to implanted devices, and it offers an appealing (but time-consuming) approach for assessing the outcomes. Rats and mice are commonly used as animal models [53, 54]. Biologists usually mark the observed behavioral patterns or the positions of the animals (e.g., sniffing, exploring, huddling, etc. for rats/mice) for frames at regular intervals that have a size fitting the task (therefore, enable the annotation but at the same time is trade-off regarding the

DOI:10.15774/PPKE.ITK.2023.008

time-resolution of the analysis). There is a wide range of available strains with differences in needs (feeding, etc.), structural or physiological parameters [55], and behavior patterns, and each study should choose the most appropriate strain model. If instances receive different medical treatments a consistent labeling is required throughout the video to analyze joint and individual behavior patterns. The need for identity-preserving tracking may arise for any strain, however, making sure that the identity labels are consistent in each frame becomes challenging even for a human annotator if instances are highly similar and lack natural markers. Although, there are several rodent marking methods such as ear tags, fur painting, fur clipping, in many experiments, none of them have been used to avoid any effect on animal behavior. In typical setups, the camera is fixed, and a homogeneous background with good contrast is used, thus segmentation of the foreground from the background is feasible.However, handling the changes in body configurations and the heavy occlusions between the instances poses a significant challenge.

Identity (ID) tracking for behavior analysis often occurs in more constrained environments, which may provide further valuable assumptions. In medical research, the environment setup aims to ensure that instances are unable to leave the scene or hide behind obstacles because proper ID re-assignment cannot be guaranteed if we lose sight of multiple instances at the same time. Such environments allow for the assumption that the number of instances in the field of view is fixed, which can be heavily exploited by the algorithmic pipeline to avoid ID switches. Using a method by which we can automatically annotate training images, implying a reduced need for human attention and time, supports the training of the models of the tracking pipeline to be easy and fast and applying them to new data if either the conditions are slightly different, or even if new animals are involved.

My goal was to create a method for high-quality segmentation of highly similar instances with unsupervised trained deep learning based on synthetically generated data that is automatically annotated to reduce the need for human effort. My research was motivated by the task of tracking two highly similar rat instances with proper

DOI:10.15774/PPKE.ITK.2023.008

preservation of the identification labels even during periods when they partially or heavily occlude each other.

I targeted a combination algorithm of a region-based detection approach [56] and contour detection that exploits inner boundaries of occluding instances, which provides a high-quality segmentation and allows reliable id-tracking using a propagation approach presented in [57]. I addressed the training of deep networks for edge detection based on synthetic data: a dataset is generated fully automatically (based on only frames with non-occluding instances), without any human annotation to train an edge detection network for detecting the separating boundary between occluding identical objects with a static background. A sample image pair is shown in Figure 1.2.

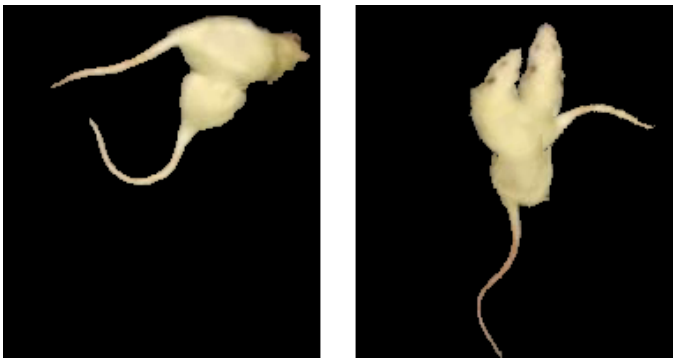

Figure 1.2: A real frame (on the left) and a synthetic frame (on the right) of two overlapping, highly similar, unmarked rats after background removal. The similar characteristics of a region of a single instance and those of the area created by the two overlapping rats pose a challenge in recognizing instance boundaries with traditional edge detection algorithms. Deep networks provide promising edge detection results and automatically annotated images that are similar to the real frames can significantly reduce the annotation required for training to facilitate a large number of biological studies, which otherwise would not be feasible.

As the "continuity" of the inner boundary edges required further improvement for precise segmentation, I was searching for a generative method, to provide an edge "completion" suited to the detected edge characteristics.

DOI:10.15774/PPKE.ITK.2023.008

DOI:10.15774/PPKE.ITK.2023.008

Chapter 2

Mapping Cellular Particle Filter to GPU architecture

2.1 Related works

There have been some former implementations to parallel architectures [33–42]. In [34], an implementation strategy is proposed which is parallel; however, it cannot maintain the local connections of the particles. The particles are split into smaller groups (around 100 particles) which perform operations independently. The information exchange among the particle groups is occasional; share ratio is suggested at around 25%. Researchers admit that the reduced flow of information of the distributed particle filter degrades the quality of estimation compared to the original algorithm which resamples according to the complete cumulative distribution.

Besides continuous information sharing, random number generation has a significant effect on the filter result. The simple solution is creating random numbers on the CPU using any of the many implemented reliable random libraries proposed in [58]. However, these researchers are aware of the great disadvantage of this technique, mainly the huge delay raised by data transfer. Hence, they prefer a sufficient GPU random sequence generator instead. In the resampling step, each particle requires information from all other particles. This adds a high computational delay. Resampling is based on the relative importance of the particles, but the technique is not restricted to uniform sweepstake over the weights (systematic resampling).

Metropolis resampler [59] in each resampling step iteratively selects B times a candidate according to a given rule for each particle. Since the aforementioned rule is based on pairwise operations, the efficient mapping on many-core architecture is

DOI:10.15774/PPKE.ITK.2023.008

realizable as follows. For each $p \in 1, \ldots, N$ particle in each $i \in 1, \ldots, B$ iteration, two main parameters, a_p^i and s_p^i, are used, where a_p^i stands for the actual particle and s_p^i stands for the selected particle; a_p^i is initialized with particle p. With uniform distribution, we draw a u_p^i random number on $[0, 1)$ and s_p^i particle from the complete particle set. If the ratio of the weights $\frac{w_{sp}^i}{w_{a_p^i}}$ is over u_p^i, the selected particle is indicated as actual. This pairwise operation can be performed independently; therefore, efficient implementation is possible on parallel architecture.

Resampling can also be accelerated if the number of particles is decreased. However, there is a trade-off between particle number and estimation accuracy. The spreading-narrowing technique in [35] proposes a solution. A tolerable N number of basis particles generates an $N \times P$ large set by propagating each particle based on the system transition model for a sequence of P states. Each P_i subset then delivers a single particle based on a local particle selection process. It either uses maximizing importance selection (MIS), taking the highest weighted particle, or it uses systematic resampling (SR) on the weights. SR has a lower complexity than a global resampling on an $N \times P$ size set as P takes values from $\{10, 20, 50, 100, 200, 500\}$ based on the current application, and for each P_i set, it can be performed parallel to each other. While MIS has an even lower complexity, it is more sensible to the noise introduced by the artificial propagative spread of the particles. For the measurements, they used a bearing-only tracking (BOT) model with 25 time steps; therefore, a direct comparison is possible for the estimation error. For [35], the position error is in the range of 0.06245 to 0.06226, which is slightly lower than our error, but still the same range. Execution time, which is the sum of sampling, weight normalization and resampling times in [35], and total runtime in our work (including memory transfers, file I/O, etc.), shall be compared to our proposed algorithm with regard to the different devices, which still indicates that our technique is faster (see Table 1).

I investigated [36–38] from CUDA ZONE, which also addresses particle filtering on GPU.However, this work is slightly out of my scope as the aim was the fast estimation of face tracking with PF; the contribution of parallelism on the GPU is relevant in the

DOI:10.15774/PPKE.ITK.2023.008

total speed-up. Three different case studies are presented for Monte Carlo methods in [36]. They found out that global resampling has a significant influence on the runtime, and they achieved 10 to 37 times speedup compared to a single threaded CPU implementation. The measurements were made with the use of a factor stochastic volatility model; therefore, direct comparison to our benchmark model is troublesome. However the computations run on the GPU, the resampling - unlike in our proposed work - is not parallel but sequential. In [37], resampling is performed with a technique based on using an offline-created and offline-uploaded texture of was single- and multiple-object tracking, using skin detection and spreading the region of interest; therefore, the lack of common model encumbers the direct comparison of the result to our proposed method.

However, in [39], the GPU device is different (making the exact time comparison difficult), and measurements were made using the BOT model [60]. The proposed particle filter method is parallel, still the random numbers are generated on the CPU, and there is no information share among local processes (e.g. in resampling). The position errors are in the same range and almost identical. Execution time in [39] is expressed as the sum of sampling, weighting, weight normalization and resampling times, whilst our presented execution time includes all operations (file I/O, memory transfers, etc). A real-world problem is presented with a particle filtering method in [40]. PF is implemented on the GPU with a distributed resampling. The work is based on sub-filters which have a limited information share among themselves. Communication can be represented as a graph where sub-filters correspond to nodes and edges are defined arbitrarily (as an attribute). Before resampling, a particle exchange step is performed among neighbouring graph nodes. However, this approach is not completely local; the amount of the exchanged particles is relatively small. Therefore, information flow is not as complete as in the final algorithm) or even as in our proposed method. Finally, we would like to mention [41] and [42] which both present parallel but non-GPU particle filters. In [41], the particles are split to subsets. Similar to [40], each subset performs the sub-steps of PF independently; however, there is no information share among the

DOI:10.15774/PPKE.ITK.2023.008

subsets. Central estimation is calculated using the results of subsets. In [42], three different techniques are presented, and locally distributed particle filter is considered as giving the best speed-up and estimation. Also, this is the closest from the three presented methods to my approach; however, operations are performed without any information share and only simulation results are given for a discrete time non-linear dynamic model of nearly constant turn. For the summary of related work, please see Table 2.1 where I highlighted the most relevant GPU-related works.

The table also reveals the difficulty of direct comparison of the results due to the different models, data and GPU devices. We can say that resampling is a key point in SIR particle filtering. Approaches to deal with SIR resampling try to optimize the speed quality trade-off for the given setup. Cellular particle filter(CPF) [45] introduces a third approach for the resampling problem by changing the logic representation of PF to a two-dimensional (2D), locally connected grid inspired by cellular neural network (CNN) architecture [46]. Each element in the grid is connected to each of its eight neighbours enabling rapid local information flow. The critical resampling step can then be performed on a subset in an r radius neighbourhood.

DOI:10.15774/PPKE.ITK.2023.008

Reference	GPU type	GPU computes	Model[a]	Number of SMs on GPU	Number of cores	GPU clock	Time includes	Runtime data
[35]	GTX 280	All; spreading-narrowing technique	BOT model 25 time steps	30	240	1.3GHz	Sampling+ weight normalization+ resampling	1 050 particles 79.4 ms, position error 0.06245; 2 000 particles 124.8 ms, position error 0.06226
[36]	GTX 280	All; sequential resampling	A factor stochastic volatility model	30	240	1.3GHz	SMC algorithm: no further details	8,192 particles 82 ms; 16 382 particles 144 ms; 65 536 particles 465 ms
[37]	8800 GTS	All; resampling uses offline-initiated texture	Skin detection + spreading region of interest	12	96	1.2GHz	Object tracking time; no further details	1.44–13.55 speed-up in fps[b] compared to CPU. Best 90 fps for multiple- and 225 fps for single-object tracking.
[38]	8800 GTX	Weight calculation	Face tracking model	16	128	1.35GHz	Face tracking algorithm time; no further details	No execution time measurements for particle filter
[39]	9400M	All; random numbers from CPU	BOT model, 25 time steps	2	16	450MHz	Sampling+ weighting+ weight normalization +resampling	For 2 048 particles: best time 168.3 ms, position error 0.078–0.083; for 4 096 particles: best time 168.0 ms, position error 0.077 - 0.081
[40]	GTX 580	All; distributed resampling	Dynamic equations to model a robotic arm	16	512	2GHz	Sum of kernels: random number generation + sampling + local sort+ global estimate + exchange + resampling	64 000 particles 0.3 ms

Table 2.1: Summary of related works including the following parameters: used model, outline of the technique and the GPU if measurements were made on it. Direct comparison is often hardly feasible due to the differences of the mentioned parameters. [a]As given in the references; [b]frames per second.

DOI:10.15774/PPKE.ITK.2023.008

Our proposed algorithm is based on cellular particle filter [45], using the idea of local neighbourhoods. Due to the CNN-type representation and the decreased dimension of resampling sets, CPF offers a solution for the problem of reduced local information change, which is stated in [34]. However, this representation is not optimal for a GPU architecture. Hence, I made some further modifications to achieve an efficient implementation which exploits the characteristics of GPUs. Besides, one of our principles is to generate random sequences on the GPU since NVIDIA SDK Mersenne Twister proved to be insufficient at low numbers. Therefore, I explored possible solutions and finally propose two different approaches for random number generation.

This section describes the necessary background and theory for hidden Markov models (HMMs), particle filters, especially cellular particle filter, and architectural perspectives.

2.2 Background and Theory

Hidden Markov Models and Particle Filtering

This section describes the necessary background and theory for hidden Markov models (HMMs), particle filters, especially cellular particle filter, and architectural perspectives.

HMMs consist of two stochastic processes. One of them is the trajectory of hidden states x_t according to $t = 0, 1, \ldots$, determined by Markov dynamics:

$$x_{t+1} = \varphi(x_t, e_1(t+1)) \tag{2.1}$$

The other contains observations y_t, for $t = 1, 2, \ldots$, depending only on the current hidden state plus an additive noise which is not limited to Gaussian.

$$y_t = \psi(x_t) + e_2(t) \tag{2.2}$$

These notable extensions transfer the resolution beyond Kalman filter [20], to the

DOI:10.15774/PPKE.ITK.2023.008

scope of particle filtering. In case of state estimation it is considered that φ, ψ, functions and distributions of $e_1(t), e_2(t)$ are given. For more information about Hidden Markov models see [19].

A particle filter is a tool for estimating the hidden states based on the observation. It is not an analytical calculation, but using a set of particles at each time step that follow the model dynamics. The algorithm is built up from three main steps in each time t (i.e. *state*).

The first step is error calculation which assigns each particle a fitness value. It is performed between the current particle value and the current observation value (same for all particles at a time step) as described in Equation 2.3. L stands for the likelihood value, for each $i = 1, \ldots, N$ particle, and l is the density function of the noise of the hidden process ($e_1(t)$).

$$L_t^i = l\left(y_t - \psi\left(x\right)\right) \tag{2.3}$$

Each particle weight is set based on this likelihood:

$$w_t^i = L_t^i \tag{2.4}$$

where L_t^i is the fitness value of the i^{th} particle, and for simplicity in resampling, each weight is normalized:

$$w_t^i = \frac{w_t^i}{\sum_{j=1}^{N} w_t^j}. \tag{2.5}$$

The second main step is resampling. While there are many alternations, I focus on a particle filter with sequential importance resampling [60]. We choose a new ξ' particle set from our current particles, $\xi'^i = \xi'^{\eta(U_i)}$, where $\eta(U_i)$ stands for the uniform random sweepstake, using the set of corresponding normalized particle weights w_t (see figure 2.1 and equation 2.6, for particles $i, j = 1, \ldots, N$).

$$P(\eta(U_i) = j) = w_t^j \tag{2.6}$$

DOI:10.15774/PPKE.ITK.2023.008

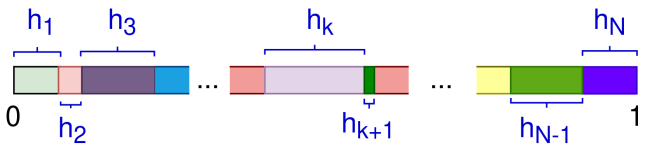

Figure 2.1: Split *[0,1]* to subintervals for resampling. Each interval has the width of the corresponding normalized weight w_i where $i \in 1 \dots N$. We generate uniform distributed random numbers for resampling, which means the bigger w_i value particle i has, the more likely it is to be picked.

The resampled set ξ'_t is used for the current estimation (e.g. taking the mean). The last main step is the iteration. In this last step we use the model to generate the next time step's initial particle set using the model:

$$\xi^i_{t+1} = \varphi(\xi^i_t, e_1(t+1)) \tag{2.7}$$

where ξ^i_t are the resampled particles, $i = 1, \dots, N$.

Particle filtering technique has been used since 1962 [61] and the SIR particle filter since 1993 [60]. Still, the proof of convergence was published only 18 years later [62]. More information about particle filters can be found in [17, 32]. Henceforward, "original algorithm" stands for the algorithm described in this section.

Cellular Particle Filter

In the resampling step, for each retake, we have to use the whole particle set. This is highly time-consuming and for a long time was considered not parallelizable. Cellular particle filter [45] offers a solution for this problem, and in contrast to other distributed particle filters [34], it maintains local connectivity, which allows for each particle to access the information of its neighbours in each time t. This ensures the same or, at some parameters, even better quality of approximation. To provide theoretical proofs for our concept is beyond this articles' scope, but see [45] for some experimental validation.

The main idea lies in the logical representation. The set of particles are organized in a CNN inspired architecture, namely, a locally connected two dimensional grid with uniform elements where each element is only connected to its 8 neighbours. Based on the

DOI:10.15774/PPKE.ITK.2023.008

connections we can define a neighbourhood for cell i (ie. cell $C_{i,j}$) with radius r: $C_{k,l} \in N_i$ if $k \in [i-r, i+r]$ and $l \in [j-r, j+r]$ (see figure 2.2).

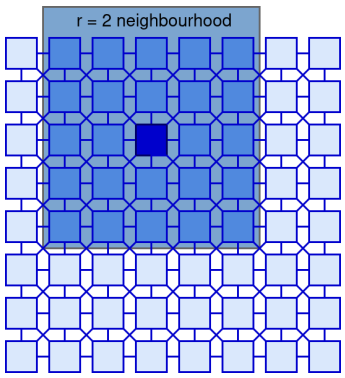

Figure 2.2: CNN processor array architecture: two dimensional fully connected grids. The light gray background highlights the $r = 2$ size neighbourhood around the black cell.

We can retrace the original algorithm if we set the neighbourhood size for each particle to fit the dimension of the grid. However, if we set a smaller r radius it defines a locally connected N_i neighbourhood for each i particle.

$$W_t^i = \sum_{j \in N_i} L_t^j \tag{2.8}$$

Using the sum of weights W_t^i of the neighbourhood, in this case the weights are set to:

$$w_t^j(i) = \frac{L_t^j}{W_t^i} \tag{2.9}$$

where $j \in N_i$.

Now, the resampling step can be performed for each i particle simultaneously within the local N_i neighbourhood according to the local distribution of the weights. Fetching the weights is realized by local communication on the physical device, therefore, it is fast. The random take on all subsets around each i particle produces N resampled particles respectively. Hence, the time-consuming part is paralellized and the required computational effort is essentially independent of the number of particles.

This method might seem to be similar to distributed particle filters [34], but the there

DOI:10.15774/PPKE.ITK.2023.008

is no communication limits among the subsets in any time states. CPF is suited to GPU architecture due to its parallel, locally connected nature. Our aim is to ensure efficient computation and therefore to exploit the properties of the GPU in our adaptation.

GPU details

Our proposed mapping is based on the GPU features summarized in this section. I used NVIDIA CUDA, see [63] for notations and details. Figure 2.3 shows the basic architecture of the GPU considered mainly from the view of mapping CPF to this architecture.

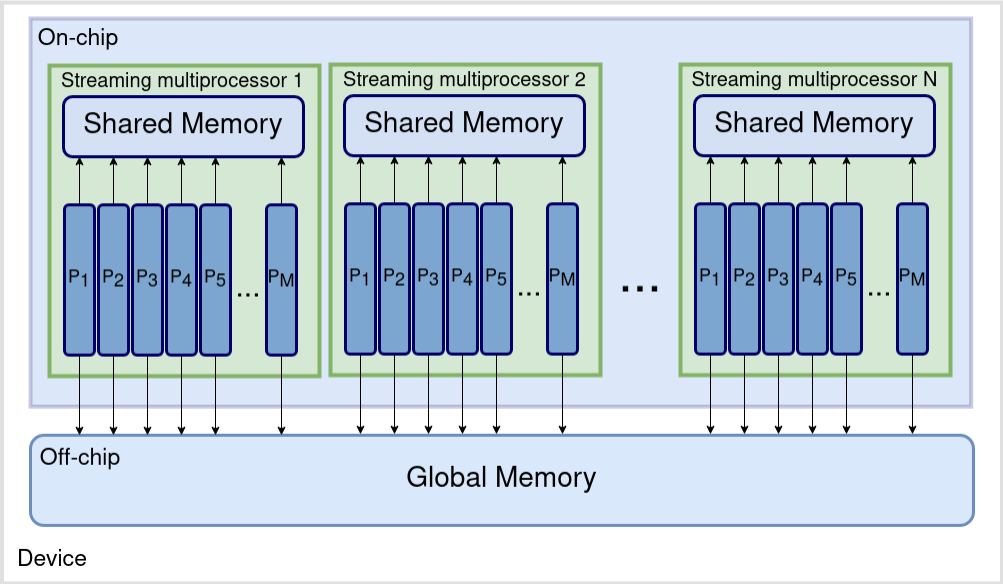

Figure 2.3: Simplified architecture of the GPU. Threads are organized in the single process elements(SPE), with an on-chip (shared), fast memory access. Besides all threads in all SPEs have access to the large and slow off-chip memory (global memory).

In the logical sense, the kernel function is the function executed on the device. It is executed simultaneously by threads. Threads are organized into blocks (typically 32 to 512 in each, based on the current task), in a one-, two-, or three-dimensional array. Blocks are organized in a grid in a one-or two-dimensional array. The number of threads per block and block per grid is called execution configuration. The thread and block organization is detailed in [63] and illustrated in Fig. 2.4.

DOI:10.15774/PPKE.ITK.2023.008

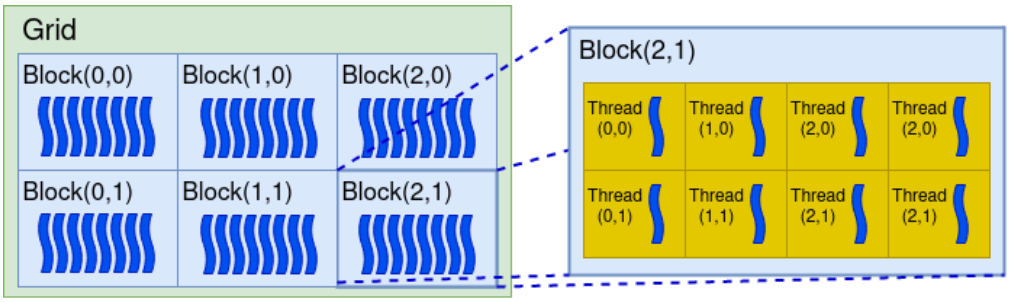

Figure 2.4: Illustration of thread and block organization, based on [63]. Threads execute the kernel functions in parallel and are grouped into 1, 2, or 3-dimensional blocks. Blocks are grouped into 1 or 2-dimensional grids.

In the physical sense, the device is built up from streaming multiprocessors (SMP). Each SMP consists of an SD RAM, a number of cuda cores and a scheduling unit. The SD RAM is an on-chip memory, with a few tens of clock cycle delays, its size is 64 KB , and it is divided to L1 cache and shared memory. The GPU has an off-chip global memory to be accessed by each SMP. Its size is usually around 1 to 4 GB, depending on the type of the particular device. Its delay is 400 to 600 clock cycles. Additionally, there are two other types of memory spaces that both reside off-chip and are cached on-chip. The first memory. The latter's size is 64 KB.

Blocks are mapped to SMPs. Shared memory of block B_i can only be accessed by the threads which reside in B_i. The communication and data share among the blocks are performed through the global memory. A fixed collection of threads is called warp. Currently, the number of threads in a warp (warp size) is 32, which is physically executed simultaneously. Besides proper memory usage, warp conflict avoidance is essential [63]. The vendor suggests block sizes multiple of the warp size to achieve the most efficient computation. However, in extreme situations, optimal block size can differ from the advised values [64]. If some threads in the warp choose different branches of operation based on the processed data, the threads within the warp may diverge. This is called warp threads which adds runtime delay. As a rule of thumb, the block size should be a multiple of 32 and each multiprocessor should execute at least six warps at the same time

DOI:10.15774/PPKE.ITK.2023.008

because the pipeline is six levels deep; therefore, $8 * 32 = 256$ may be an ideal thread number.

To achieve a high throughput, on-chip memory (shared memory) should be used if threads require frequent data access. Therefore, the main computational tasks are performed block-wise, in the shared memory of the blocks, and only necessary global synchronization is performed through global memory. Although the two-dimensional texture and surface memory would also be feasible, shared memory throughput can be optimized for a one-dimensional array type representation; thus, structural aspects in CPF algorithm were reconsidered.

In the shared memory, I use arrays with the size 512 for the particles, the error terms, the normalizing sums and the uniform random numbers. In our case study, when taking the highest neighbourhood size and at a single precision 10 250 bytes are occupied. Each shared memory can access a 48 KB memory/multiprocessor at compute capability *2.x* [63]. Global memory in recent GPUs is at least 1 GB which restricts the number of states in the observation and estimation, but both memory access is high enough for our computations.

2.3 Methods

Random number generation

The SIR in each t time step requires the same amount of random numbers as the number of particles. I intend to generate these random sequences on the GPU device instead of the CPU to spare repetitive data transfer between the main memory of the system and the global memory of the device as recognized in [58]. The distribution of the random numbers in the resampling is critical on the quality of the estimation. If it is not uniform, though the drawing of particles should depend on the weights exclusively, then it would be biased.

Recently GPU random number generation for various purposes has become widely investigated and well tested (see [65–67]). NVIDIA provides two solutions for random

DOI:10.15774/PPKE.ITK.2023.008

number generation. The first option was the Mersenne Twister in the SDK. Unfortunately, I observed that the generated distribution is inappropriate for a small set (hundreds or thousands) of numbers and is primarily admissible for around two million numbers and above. I made delicate modifications to get admissible distribution (see B for details). The second option was curand, which proved to be fast and appropriate.

GPU CPF algorithm

As described at CPF subsection, particles are represented as arranged on a 2D grid with local connections and have a given neighbourhood radius. Although GPUs have a different kind of architectural organization compared to the CPF structure, it is possible to map the 2D topology to GPUs' memory hierarchy. However, two modifications were made in the particle topology to fit better to the architectural details of GPUs, namely, instead of 2D, a 1D topology was applied and the neighbourhood was considered circular (see Figure 2.5) in one direction. There were two reasons for these decisions. First, the proportional size of the neighbourhood is smaller in the 1D case. Second, using only one side of the neighbourhood, the coalesced memory access is ensured.

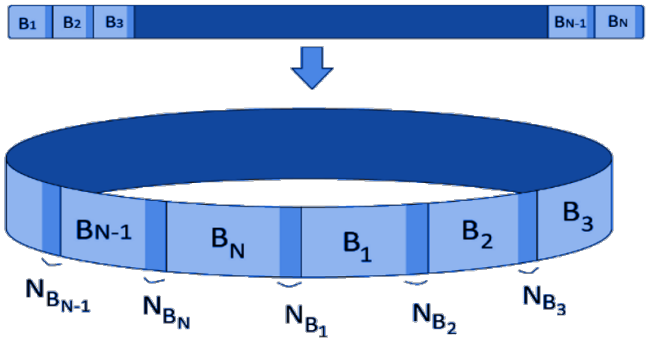

Figure 2.5: Restructuring linear representation of N blocks to a ring type topology. B_i stands for the i^{th} block of threads, and N_{B_i} for the corresponding neighbourhood from the previous block, $i \in 1, \ldots, N$.

The local connectivity is preserved by choosing each shared memory array size higher than the thread number exactly with the size of the required neighbourhood. Each thread with index i can obtain its k neighbours at indexes $i-1, i-2, \ldots i-k$.

DOI:10.15774/PPKE.ITK.2023.008

In the following I specify some implementation details. Although the operations are mainly performed in the shared memory because of the synchronization and CPU–GPU data transfer, the following variables are global memory arrays: the observation sequence (Y), the state estimation (X) and the set of particles $x_{particles}$. The number of particles is denoted by N. Y is naturally given, X is empty and $x_{particles}$ is initialized with N samples of the same distribution as n_t described in equation 2.10. The number of threads in a block was set to 256. The size of the neighbourhood is r, meaning each particle is connected to exactly $r + 1$ particle (every particle is connected to itself). In each block two shared memory arrays of size $256 + r$ are created for particle states x_{shared} and fitness values L_{shared}; and additionally, two arrays whose size equal the number of threads in a block are allocated for uniform pseudo-random numbers U_{shared} and normalizing weights w_{shared}.

In each time step I copy the particle values from the global memory to the shared memory by overlapping split (see Figure 2.6 for illustration).

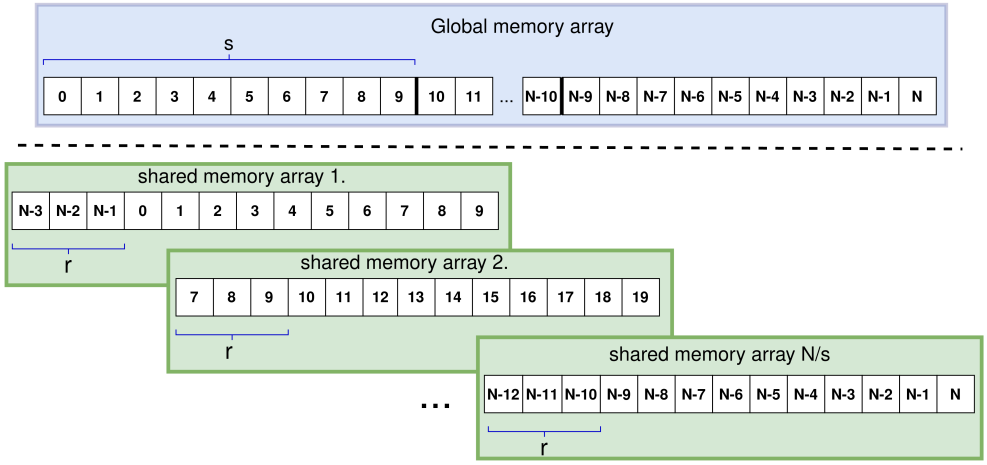

Figure 2.6: Splitting a global memory data array to shared memory keeping local connectivity, where 's' stands for the number of a threads in each block, and r for the size of neighbourhood. To fit the architectural details of the GPU and reduce computational time I applied 1D topology instead of the proposed 2D grid with one sided neighbourhood.

I load 256 values respectively to each shared memory to the particle's array (x_{shared}), but sparing the first r elements of it. These positions are filled with the r neighbours

DOI:10.15774/PPKE.ITK.2023.008

in the global memory of the first element in x_{shared}. For the very first element, I use a circular approach by taking the values from the end of the global memory array.

There are three kernel calls for each estimated values to provide full synchronization. The main kernel performs the following operations in each time step t (also see Figure 2.7), where global and local thread IDs are defined as follows: $i_{gl} = blockDim.x * blockSize.x + threadId.x$ and $i_{loc} = threadId.x$, where $threadId.x$ is the thread index in the thead block, $blockSize.x$ is the number of thread per each block, and $blockDim.x$ is the index of the block. For more information about terminology see [63].

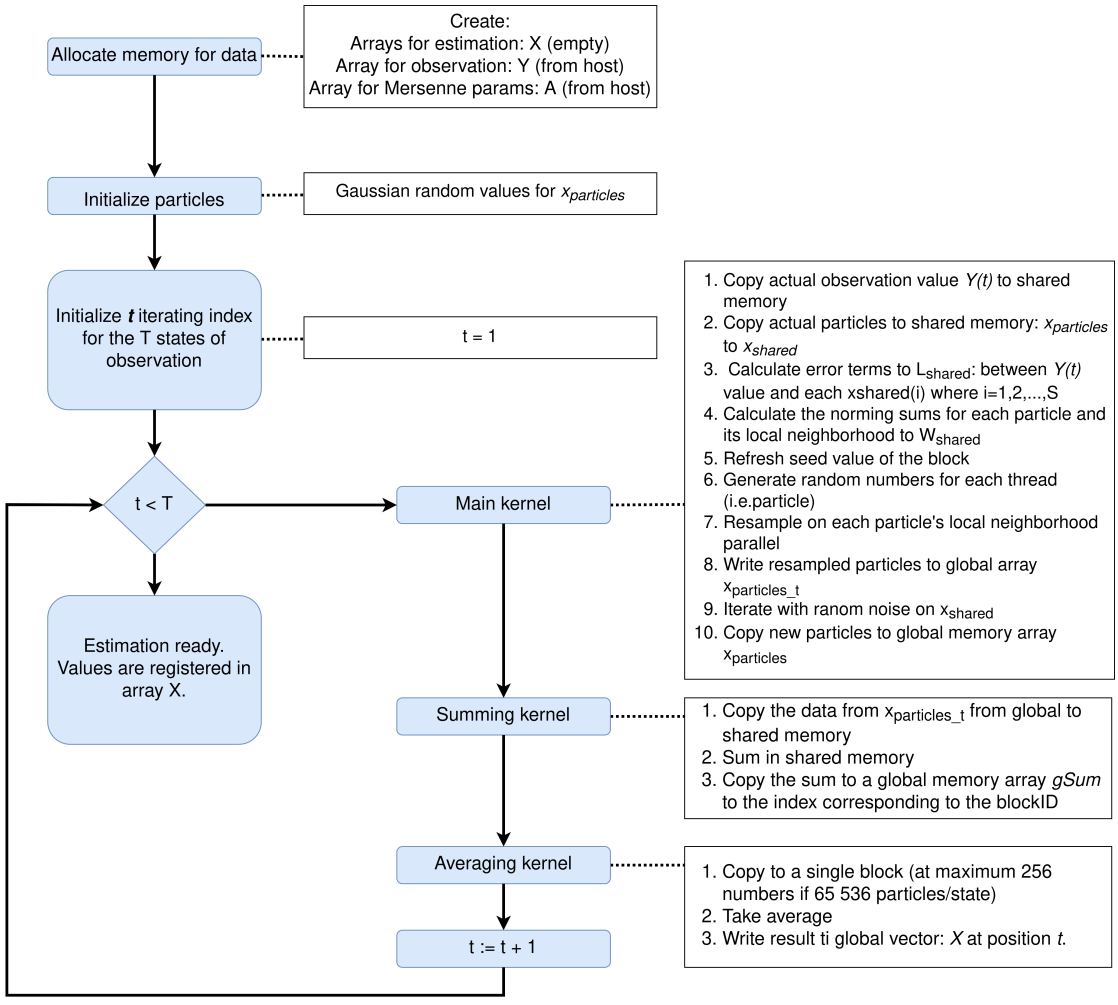

Figure 2.7: Flowchart of our implementation on the GPU device. S denotes the number of threads in each block, $x_{particlec_t}$ denotes the resampled particle set in t state. The observation sequence consists about T states.

DOI:10.15774/PPKE.ITK.2023.008

I. Initialization

2. $x_{shared}[i_{loc}+r] \leftarrow x_{particles}[i_{gl}]$
2. if $i_{loc} < r$ then $x_{shared}[i_{loc}] \leftarrow x_{particles}[i_{gl}-r+i_{loc}]$
2. if $i_{gl} < r$ then $x_{shared}[i_{loc}] \leftarrow x_{particles}[N-r+i_{gl}]$

II. Error calculation

3. $L_{shared}[i_{loc}+r] \leftarrow l(Y[t]-x_{shared}[i_{loc}+r])$
3. if $i_{loc} < r$ then $L_{shared}[i_{loc}] \leftarrow l(Y[t]-x^s[i_{loc}])$
4. $w_{shared}[i_{loc}] \leftarrow L_{shared}[i_{loc}]+\cdots+L_{shared}[i_{loc}+r]$ get normalization sums for each particle

III. Resampling

5. if $i_{loc} == 0$ refresh seed value of the block
6. fill U_{shared} with uniform random numbers
7. Iteratively sum $L_{shared}[j_{loc}]/w_{shared}[j_{loc}]$ where $j_{loc} = i_{loc}, i_{loc}-1, \cdots, i_{loc}-r$. Stop if adding a $L_{shared}[k_{loc}]/w_{shared}[k_{loc}]$ term affects the sum to exceed $U_{shared}[i_{loc}]$ for the first time.
8. From the neighbourhood of i_{loc} the corresponding k particle is selected to $x_{particles_t}[i_{loc}]$.

IV. Iteration on particles

9. fill U_{shared} with uniform random numbers
9. apply Box-Muller transform on pairs of uniform samples (n_t, stored in thread level)
9. $x_{shared}[r+i_{loc}] \leftarrow \varphi(x_{shared}[r+i_{loc}], n_t)$
10. $x_{particles}[i_{gl}] \leftarrow x_{shared}[i_{loc}+r]$

DOI:10.15774/PPKE.ITK.2023.008

The estimation is performed by two kernel calls. The first kernel calculates the sum for each shared memory $x_{particles_t}$ arrays to a global memory array $gSum$. The second kernel takes the average value of $gSum$ with respect to the number of particles.

Besides, the following optimization techniques were used to achieve optimal efficiency on the GPU: (1) random number generation, resampling, average calculation, Gaussian transform for the iteration operations are performed only on the relevant part of the shared memory to spare time; (2) the number of *if* statements is minimized as possible, and are transformed to ternary expressions; (3) the shared memory arrays are reused, therefore even some parameter passes can be spared.

2.4 Evaluation and results

Model

The implemented algorithm was tested on two different widely used models. The first was the following benchmark model [17, 68–70]:

$$x_{t+1} = \frac{x_t}{2} + \frac{25x_t}{1+x_t^2} + 8\cos(1.2t) + n_t \tag{2.10}$$

$$y_t = \frac{x_t^2}{20} + u_t \tag{2.11}$$

This model is non-autonomous, non-linear and has a continuous state space, thus linear tools for state estimation are not applicable. The state of the system is x_t, the observation is y_t; n_t and u_t are IID Gaussian sequences, $n_t \sim N(0,10)$ and $u_t \sim N(0,1)$.

The second model was a bearings-only tracking (BOT) model originally presented in [60] and also analyzed in [39, 40]. For the illustration about the trajectories of each model, see Figures 2.8 and 2.9.

Measurements

There are two aspects of the measurements, namely, the average quality and the required time for one estimation. These two quantities were monitored with different

DOI:10.15774/PPKE.ITK.2023.008

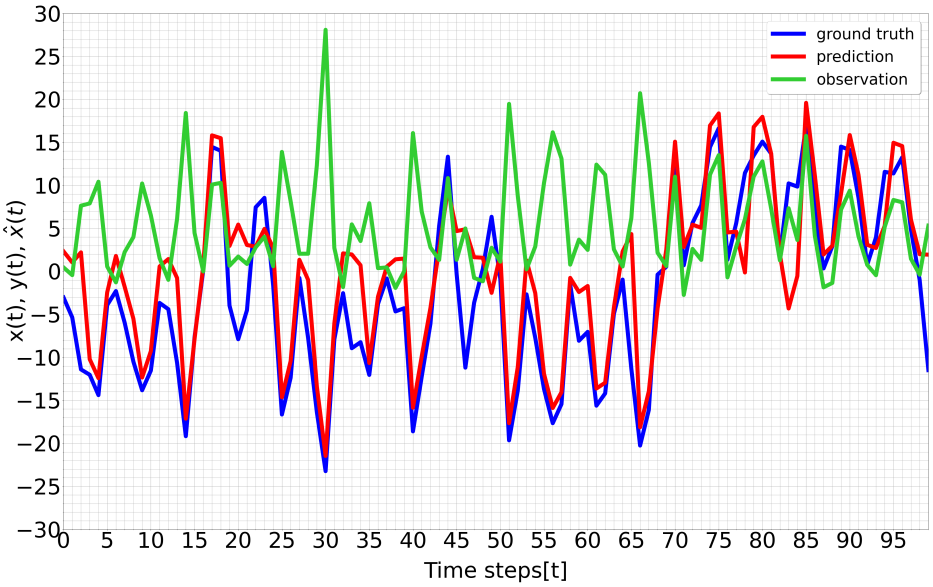

Figure 2.8: Trajectory for the first benchmark model. The hidden states are marked with blue and the observation values of the states are marked with green. The state estimation of our GPU CPF is marked with red.

configurations (number of particles and radius of neighbourhood) of the filter. The number of particles were swept through the following values: 2 048; 4 096; 8 192; and 16 384, while the radius of neighbourhood took the following values: 32; 64; 128, and 256.

Consequently, 24 pairs of these are composed; in our terminology, these are called configurations (i.e. N, r pairs). For the first model, the input observation trajectories (y_t) during the tests were exactly the same as the ones used in [45] to ensure a fair comparison. For the BOT model, I generated 100 trajectories over 24 time steps based on the given state transition equations in [60] and, respectively, the observations.

Measurements were done on a PC with Intel i5-660@3.3GHz ($ 4MB), 2 CPU core with 4 GB RAM running Ubuntu Linux 11.04 with kernel version 2.6.38-15 (amd64). I used an NVIDIA GeForce GTX 550 Ti GPU with 1 GB GDDR memory with CUDA toolkit 4.1 with 295.49 driver version[1]. The following NVCC compiler options were used to drive the GPU binary code generation: -arch=sm_20; -use_fast_math. I also

[1] Available at the time of the research in 2013.

DOI:10.15774/PPKE.ITK.2023.008

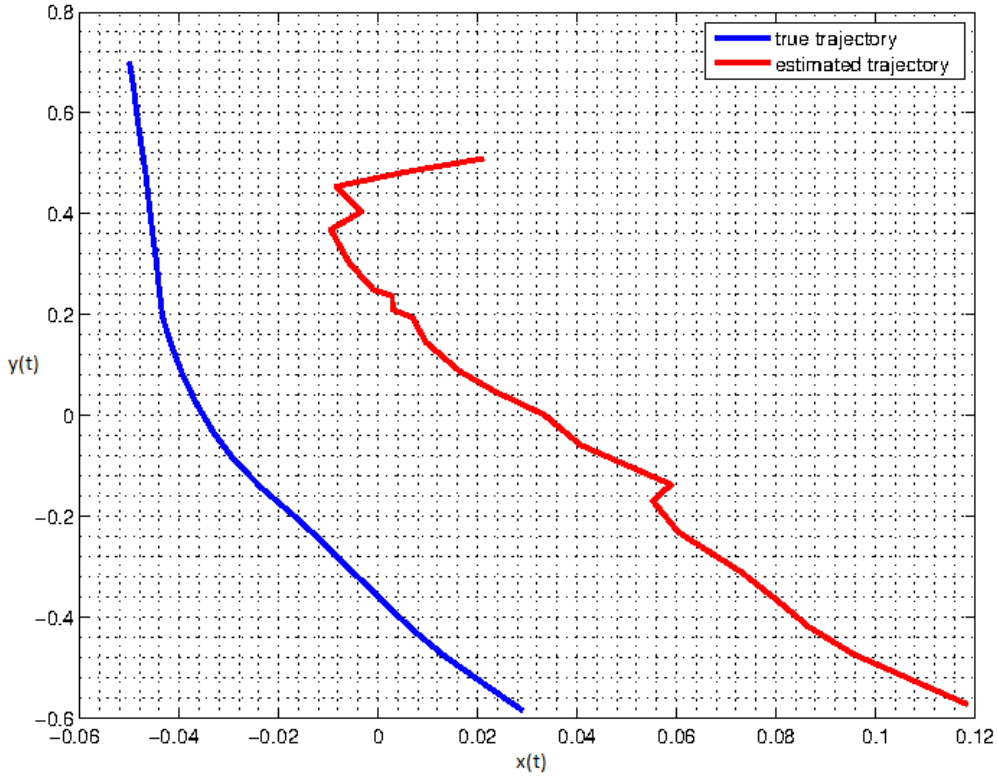

Figure 2.9: Trajectory for the BOT model. The positions ($x-y$ cartesian coordinates) are the hidden states (blue), the estimation from the GPU CPF is red. The mean of the position error is about 0.08.

made some measurements with -arch=sm_13. The sm_13 and sm_20 options in CUDA C programming refer to specific compute capabilities, namely compute capability 1.3 and 2.0 respectively. These options are used during compilation to take advantage of the capabilities provided by the corresponding compute architectures. On the other hand, the fast math option, when enabled, can result in faster execution of certain mathematical functions such as division and power operations. However, it's important to note that enabling the fast math option may introduce some trade-offs in terms of accuracy. For more details see [63]. The host C code was compiled with GCC 4.5; the compiler optimization flag was -O2. GPU kernel running times were measured with the official profiler provided by the toolkit, and the global times were measured by the OS's own timer. The kernel time measurements include the particle filtering kernel of a single

DOI:10.15774/PPKE.ITK.2023.008

time step; the global times include all operations during the execution for all states (file I/O, memory allocations, computational operations, etc.).

Estimation quality

The quality of estimation for the first model was measured by the MSE between each hidden and estimated trajectories, for the BOT model by the position error (ie. Euclidean distance). I used 1 000 and 100 different state sequences (i.e. observation sequences) for each configuration in the first model and the BOT model, respectively.

Figure 2.10 presents the quality of estimation for the first model; Figure 2.11, for the BOT model, namely the measurement error with respect to the measurement time. Each point represents a configuration since its x and y coordinate values are the mean of 1 000 and 100 executions for the two models, respectively. For the first model, it can be seen that using more than 4 096 particles slightly improves the quality of estimation.

However, for the BOT model estimations, where the particle number is more or equal to 2 048 (alike [34, 39]), provide a fair result. The position error is in the same range as in [34, 35, 39] and our proposed method. The results suggest that the proportion of the neighbourhood size to the particle number realizes an information sharing ratio among the particles. This can be seen in Figure 2.11: the optimal ratio for the configuration is when the position error is minimal, typically marked with squares except 2 048 particles when marked with triangle.

Time

Figure 2.12 presents the total runtime of the kernels in the first model. The blue lines represent the running times with compiler option -arch=sm_13; the red lines, with -arch=sm_20; fast_math.

Kernel runtimes with different nvcc flags. This figure shows the difference between the total running times for the different compiler options for the first model. The use of fast math and sm_20 has a significant effect on the kernel times.

The first compilation setting will be referred to as old target code; the latter, as new

DOI:10.15774/PPKE.ITK.2023.008

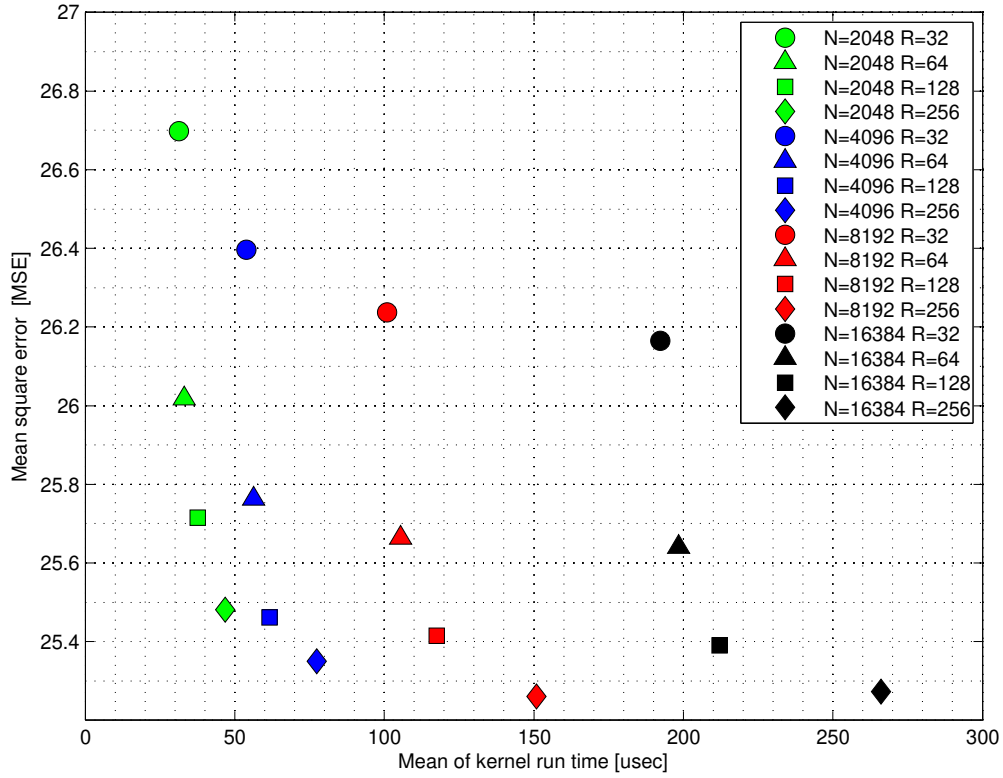

Figure 2.10: Estimation quality for the first model. This figure shows the mean square error of the estimation as a function of kernel times. It can be seen that at a given particle number with the increase of the neighbourhood size, the estimation quality improves simultaneously. N stands for the number of particles, R for the size of the neighbourhood.

target code. It can be seen that for the neighbourhood size below 64, the old target code performs 20% faster than the new. With the new target code, I can achieve a 40% to 45% improvement in execution time if the old target code is considered as 100%.

Due to the logic of physical mapping of blocks to multiprocessors, the GPU is under-utilized for particle numbers under 2 048. Above this particle number, the scaling of the execution time is nicely illustrated in Figure 2.12.

For various neighbourhood sizes, we can say that the required time is proportionally increasing to the number of R. This phenomenon is due to the resampling step as it examines the candidates sequentially for resampling. Even if the proper particle is found, the loop does not terminate until the current particle is compared to all of its neighbours to avoid warp desynchronization.

DOI:10.15774/PPKE.ITK.2023.008

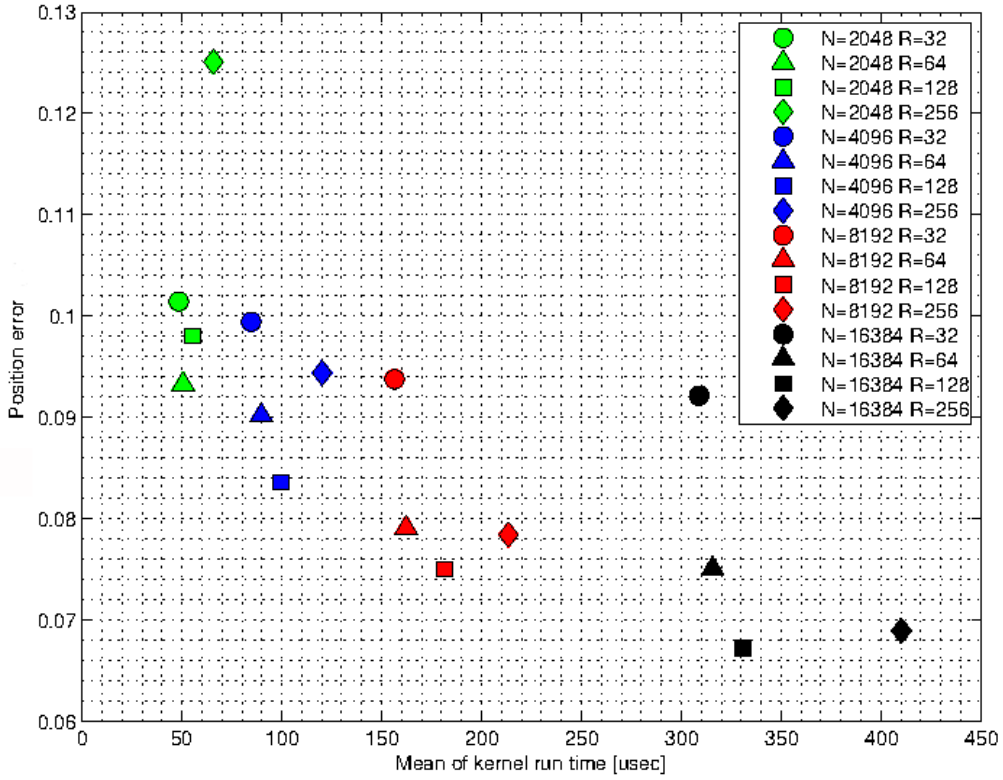

Figure 2.11: Estimation quality for the BOT model. This figure shows the position error of the estimation as a function of the kernel times. For this model, it can be seen that there is an optimal information share ratio where the position error is the lowest for a given particle number at a neighbourhood size.

The execution time of particle filter (including file I/O, initialization of random number generation, memory transfer between CPU and GPU, etc.) is 77 ms for the BOT model. However, I used the same model as in [35, 39], an exact comparison is hardly available due to the differences of the GPUs and it is not specified what their time measurements include. For the first model, see total execution times with and without host code optimization (made by compiler) in Figure 2.13.

Discussion

A key point of our proposed algorithm is the local resampling technique which has a high influence on the estimation quality. This key point can be viewed as diffusion of information inasmuch as every particle with relatively high likelihood attracts all

DOI:10.15774/PPKE.ITK.2023.008

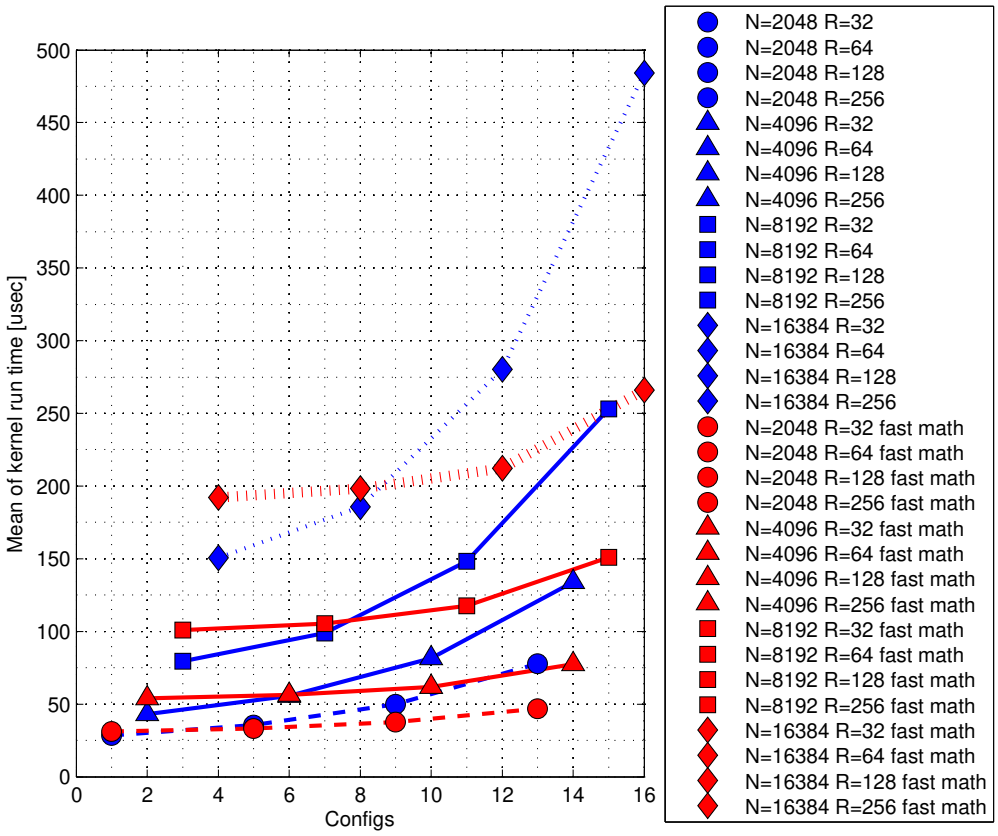

Figure 2.12: Kernel runtimes with different nvcc flags. This figure shows the difference between the total running times for the different compiler options for the first model. The use of fast math and sm_20 has a significant effect on the kernel times.

particles which has this likely particle in its neighbourhood. In this way, the other particles not having this very particle in their neighbourhood are not affected in this state estimation time step. Although it is not the traditional full resampling, it enables the algorithm to be sufficient even at high-uncertainty dynamic models. The BOT model is not a highly uncertain model as sharp changes are unlikely, but in the one dimension benchmark model (as you can see in Figure 7), rapid and significant changes are typical. Cellular resampling preserves the diversity of the particles to avoid quality loss. The estimation error for a given particle number changes within a narrow range around the optimal, depending on the neighbourhood size. This modulation is not in direct or inverse ratio to the number of used neighbours but follows a descending and then

DOI:10.15774/PPKE.ITK.2023.008

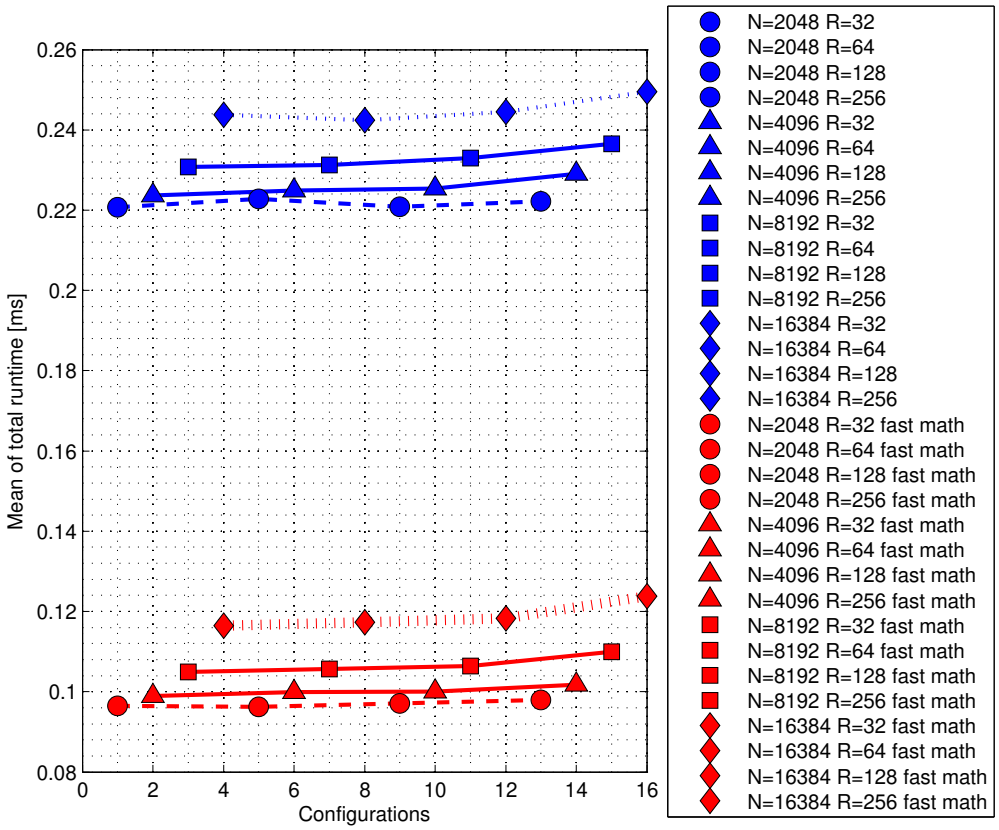

Figure 2.13: Global runtimes for the first model with and without O2. This figure shows the speed-up of the host code optimization compiler option for the first model.

increasing characteristic (like the shape of letter 'U', see Figure 10). This indicates that for a given model, at any particle number, there exists an optimal share ratio range among particles to achieve the lowest error. In our proposed method, the information sharing ratio is tunable and may be modulated adaptively; therefore, it broadens the range of options than using a predefined information share value. For further details and reasoning, see [45].

To ensure the local diffuse information share, I used shared memory arrays. Due to the high number of writing and reading data arrays (particle samples, weights, likelihoods, resampled values etc), if it were performed in global memory alone, the performance would be worse. However, the use of global memory cannot be totally evaded as synchronization and regular information sharing among blocks are essential.

DOI:10.15774/PPKE.ITK.2023.008

Without this synchronization, a similar information loss and quality degradation would appear as in the case of the distributed particle filter presented in [34] though this synchronization is a time quality trade-off. However, using constant memory (which is cached) would expose an attainable solution; only the observation values could be stored in the constant memory since all the other values are generated during kernel execution. Additionally, it would not improve the performance as reading from the constant memory space requires a fetch from the off-chip memory to cache the value of the current observation (like the current fetch from the global memory to the shared memory), and the access time of the constant cache is similar to the access time of the shared memory.

Two different models were used in this work. The first one is a synthetic benchmark model. It does not model any physical system of practical interest. It is just a widely used highly non-linear model since both the observation and the state transition is non-linear, unlike the other model (BOT model) which has a linear state transition. The Root Sum of Squares (RSS) for the proposed method with 1 024 particles is 50.52. This value falls within the same range as the errors reported in [34] and [45]. For 900 particles, the RSS values were 50.60 for [34], 47.01 for the CPF emulation on a 32-bit architecture in [45], and 62.10 for the CPF implementation on a Xenon architecture in [45]. This second model describes a bearing-only tracking of an object in the two-dimensional $(x - y)$ plane, with a fixed observer position, where observation (z) is the bearing of the object trajectory. Through the BOT model, I can compare the estimation quality of CPF to GPU particle filters in [34, 35, 39]. We can see that the error is in the same range with [35, 39] and is better than the error in [34]. Comparison is presented in Table 2.2. According to the error and time measurements, we can state that this is a feasible mapping from a virtual machine (CNN UM inspired architecture) to a state-of-the-art architecture (as of the time of this research in 2013) with a mature ecosystem available at a low cost.

DOI:10.15774/PPKE.ITK.2023.008

Table 2.2: GPU CPF estimation quality comparison with state-of-the-art methods (as of the time of this research in 2013) for the BOT model. For each approach, the Mean Square Error (MSE) for the y-coordinate or the position error is compared (as available). N stands for the number of particles.

BOT model estimation quality		
Approach	**N**	**MSE**
[34]	4 096	0.009 – 0.01
GPU CPF	4 096	0.0095
	16 384	0.0064
Approach	**N**	**Position error**
[35]	1 050	0.06245
	2 000	0.06226
GPU CPF	2 048	0.0932
[39]	2 048	0.078 – 0.083
	4 096	0.077 – 0.081
GPU CPF	16 384	0.06895

In the GPU adapted CPF algorithm, each thread executes roughly 300 to 2 100 floating point operations (at double precision). This depends on the neighbourhood size. Those operations which are performed through the neighbourhood are additions and unfortunately divisions (calculating the actual weights with the norming sums in the resampling). This amount of divisions are clearly one bottleneck. The speed-up achieved with fast_math also supports this explanation. The other bottleneck of the algorithm is the shared memory size and access pattern. If more threads could reside in a block, then the ratio of the overlay among blocks due to the neighbourhood size would be less. In the resampling, branches are unavoidable, and fork and join of threads within a warp are necessary. There are a number of for-loops which iterate through the given number of iterations where the end index is unknown at the time of host code compilation. In our framework, it is only known in runtime. Still, for a given application, optimal parameters can be set directly in the code based on measurements,

DOI:10.15774/PPKE.ITK.2023.008

and therefore, loop unroll can be applied. Another approach can be just-in-time (JIT) kernel compilation. This method is effective only if the JIT compilation time is less than the cumulative gain from the loop unrolling through the state predictions.

On the one hand, if we would perform the algorithm as purely sequential, the order would be $O(N \times R)$ without the random number generation which (as mentioned) is not part of the basic task. On the other hand, with a virtual GPU on which all blocks are active simultaneously (totally utilized), the order of the algorithm would be $O(R)$ as the threads are independent from each other, not counting the synchronization points. However, on GTX 550, the time increase starts after 1 536 threads (on 6SMPs$\times$ 256 threads). On GTX 580, there are 16 multiprocessors in comparison with the 6 multiprocessors of GTX 550 ; therefore, under 4 096 threads, the GPU would be under-utilized (i.e. GTX 580 is the top GPU of the Fermi GPUs).

A direct comparison to a CPU, sequential particle filter, is not entirely adequate; still, I would like to mention differences in the running time. In [45], particle filter was realized and used as a reference for time measurements of the proposed algorithm; therefore, I compare our results to this also. However, time measurements are not presented for many particle numbers; measurements of 4 096 particles already indicates the benefits of our implementation compared to the CPU version. Execution time of the algorithm was 16.417 s for the first model on a dual-core processor PC (Intel T6570) with 2.1GHz. Compared to our 100 ms execution time (including file I/O operations between the CPU and GPU, on a NVIDIA GeForce GTX 550 Ti GPU), we can say that a 164x speedup is achieved.

Conclusions

I introduced the first adaptation of CPF to GPU architecture. Compared to [45], I measured the performance on a real hardware. The strength of this approach is to maintain the local connectivity to prevent information loss, while the position error and the execution time are comparable to those of [39] if we assume that their measurements also include all operations (file I/O, random number generation, etc.). The algorithm

DOI:10.15774/PPKE.ITK.2023.008

utilizes architectural features: whilst CPF algorithm would demand two-dimensional representation, (e.g. texture or surface), I modified the algorithm to enable one-dimensional processing and still kept the local connectivity and the local neighbourhood-based reduced and parallel processing.

The compute capability of the GPU determines the maxima of the number of threads that can be handled at each state. The sequential operations are performed in the shared memory thus shall make no effect on running time when we increase the number of particles. Still, as the shared memory blocks are arranged to multiprocessors as defined on the device, there is a scheduling which introduces a delay. Therefore, however parallel, the algorithm still will require more time at more particles.

The proposed method is independent from the random generator if the quality of the uniform and normal distribution is acceptable. As the first approach, I used NVIDIA Mersenne Twister for which the corrections to achieve adequate distribution, see details in the Appendix. In the second approach, I used curand which generates appropriate distribution quickly. This adaptation of CPF by delicate modifications presents GPU as an excellent platform to solve problems that could not be solved real-time previously.

DOI:10.15774/PPKE.ITK.2023.008

Chapter 3

Automated multi-animal tracking for highly similar rat instances

3.1 Related Works

Multiple methods address the problem of processing visual data of biomedical research of rodents, with computer vision. Deep Ethogram [71] identifies social interaction types, but cannot track identities; therefore, no individual patterns are available. DeepLabCut [72] utilizes keypoints and tracklet stitching methods in multi-object pose estimation and tracking. For a demonstration of identity prediction and tracking, they use a dataset of marmosets, with two animals in a cage, and a light blue dye is applied to the tuft of one of them, which is a distinctive visual marker. Moreover, for training, they used 7 600 human-annotated frames, each containing two animals and 15 keypoints per animal. Currently, DeepLabCut has no pre-trained model available on rodents.

Diverse approaches [73–75] are based on a module of DeepLabCut utilizing heavily prior human annotations of several frames, with multiple keypoints and keypoint connecting edges per animal. SLEAP uses the so-called human-in-the-loop method to achieve the necessary accuracy for pose estimation over the frames, which becomes continuous annotation in the prediction phase.

Idtracker, idtracker.ai, and ToxID [76–78] do not require prior data annotation; therefore, they are similar approaches to our proposed pipeline. However, for idtracker.ai the used C57BL/6J mice have a homogeneous black color, except their ears, which are visible in all poses, therefore representing a strong visual marker that may be

DOI:10.15774/PPKE.ITK.2023.008

utilized in feature learning of deep networks. On the other hand, idtracker.ai requires that the areas of the individuals barely change, a severe restriction for segmentation and separation of non-shape preserving instances. Rats are flexible and may take different forms in 3D, e.g., during an interaction or rearing [79] that introduces a large range of pixel-wise size variations.

I applied idtracker, idtracker.ai, and ToxID on our data with various parameter settings. I found that several ID switches occur even for relatively simple videos. I show two such switches in Figure 3.1 for idtracker.

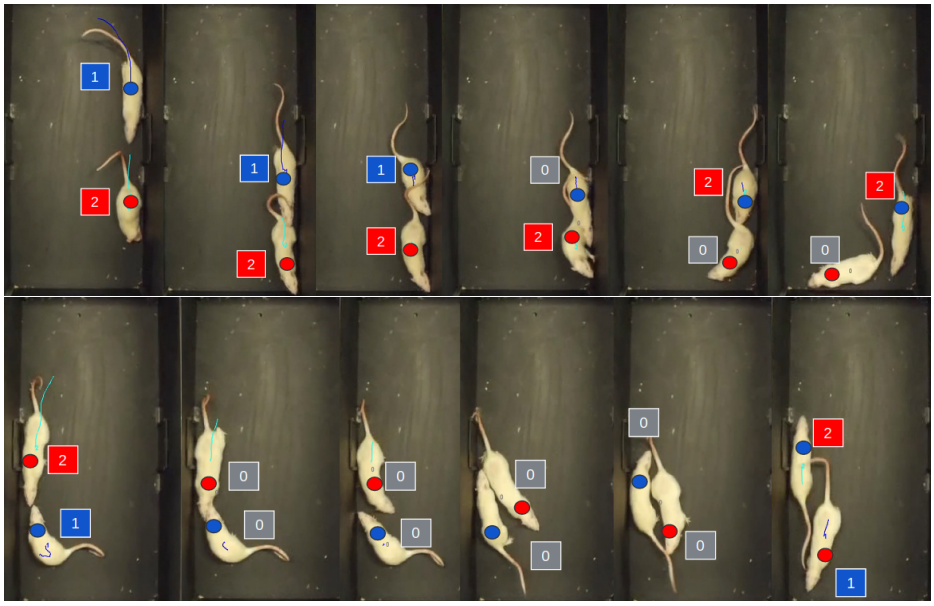

Figure 3.1: Two track switching examples using "idtracker" on our test data (cropped from the frames of the video). Ground truth labels are colored disks: blue (label 1), and red (label 2). Predicted labels are within the squares. During occlusion, frame labels are lost (label 0), and IDs and ground truth labels differ.

ToxID method [78] segments individuals from the background and stitches traces across all frames. ToxID needs optimal illumination conditions, and along with idtracker.ai, it utilizes the fact that body shape and appearance do not change much. In addition, the similarity-based probabilistic texture analysis is not reliable for videos longer than about 20 min [80]. For ToxID an implementation is available, called ToxTrac [81]. I show a segmentation on a sample from our videos in Figure 3.2. ToxTrac

DOI:10.15774/PPKE.ITK.2023.008

does not aim to track the individuals without identity switches, but all tracks are saved for further statistical analysis, which would possibly be useful for joint behavior analysis.

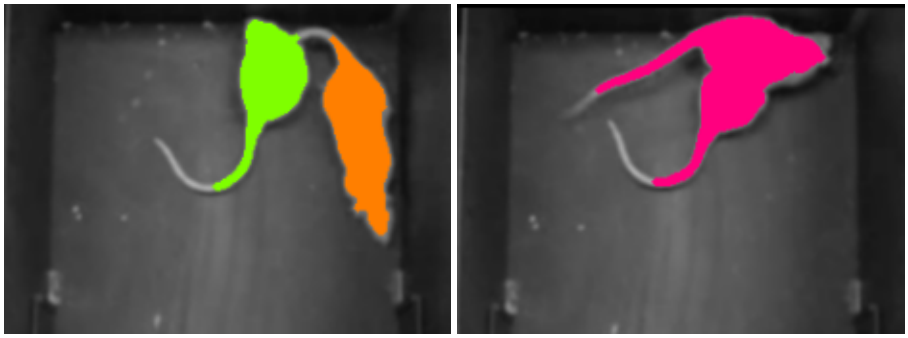

Figure 3.2: ToxTrac, which requires no preliminary annotation, was applied on our test sequences. Non-occlusion instances are usually well detected, but instances are not separated during occlusions: a single mask is provided.

The method in Lv et al. [82] is built on YOLO and Kalman filter-based prediction for tracking bounding boxes of multiple identical objects (three white rats). In this method, human annotation is reduced by training on an automatically pre-calculated annotation of 500 frames, and requiring only some correction. However, the training dataset focuses mainly on non-occluding positions and the model is not prepared to track identities over episodes of significant overlaps. Object ID-s are switched often in the video (accessed on 16 October 2021, CC BY 4.0) provided by the authors, see a small illustration of a few frames in Figure 3.3.

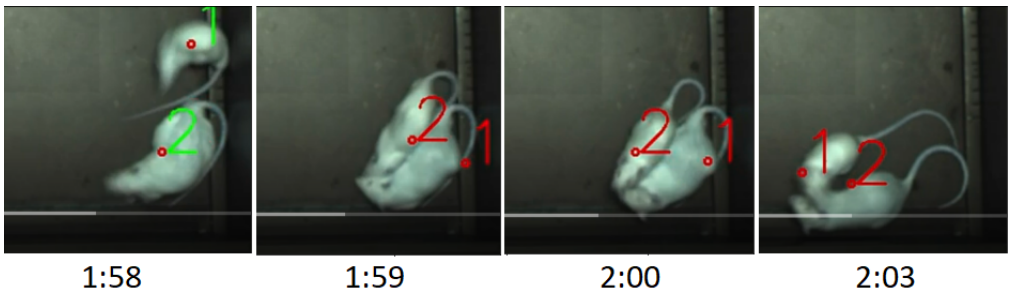

Figure 3.3: Some examples for frames selected from a demonstration video attached to [82] where ID tracking fails. The numbers 1 and 2 correspond to instance IDs. During episodes of significant overlaps multiple ID changes occur.

DOI:10.15774/PPKE.ITK.2023.008

In the past few years, there has been high interest and extensive research in deep networks [72, 83–89] to improve instance segmentation over the combination of traditional image processing methods. When instances are highly similar and have few or no features and similar colors, regions are of little help, whereas edges still may be detected. In my research, I utilize deep learning-based edge enhancement for instance segmentation.

When applied to complex scenes with multiple objects, basic edge detection methods such as Sobel [90] or Canny [91] use threshold values as parameters for balancing between being detailed enough, but not too noisy. When it comes to highly similar and occluding objects, these methods fail to detect crucial edges for instance separation.

There have been various attempts in deep learning instance segmentation under occlusion. Lazarow [92] proposed to build upon a Mask R-CNN [84] based architecture, but utilizes the differences of the objects. This approach achieves good results on benchmarks but fails to differentiate highly similar instances, such as the task of tracking rats.

In the case of highly similar and occluded objects, instance separation is often not possible using regional information only. When dealing with instances that have the same visual appearance (e.g., two white rats) edge information can help the segmentation of the overlapping instances. The deep DexiNed network [93] provides promising results for both inner and external edges. This architecture is based on the holistically-nested edge detection (HED) scheme [94], and with the fusion of the network layers, the produced output balances low and high-level edges. DexiNed was trained on the Barcelona Images for Perceptual Edge Detection (BIPED) database [93]. DexiNed predicts well-defined outer boundaries for the instances, but the inner edges in overlapping positions are discontinuous. Deep generative networks can be exploited in such situations, that are capable of filling in the missing edge fractions. For filling in the missing parts of the edges inside the overlapping instances, I was inspired by the edge generation network called Edge-Connect [95]. Edge-Connect uses two GANs in tandem. It masks the images and trains an edge generator to estimate the edges in the masked regions. The second stage

DOI:10.15774/PPKE.ITK.2023.008

of the tandem learns the image completion task building upon the structure information of edges detected in the first stage. For our purpose, the first stage of this double GAN is useful as our interest is in the separation of the animals. In the case of highly similar and occluded objects, accurate boundary information is hard to acquire.

Additional regional information such as the recognition of body parts may be invoked to find missing edges and to decrease the errors. Kopácsi et al. [56] proposed a method to detect body parts of rodents without the need for hand-annotated data. They used (i) various computer vision-based methods to find keypoints and body parts from foreground-background masks, (ii) synthetic data generation, and trained a Mask R-CNN [96] for keypoint detection and body part segmentation on unseen images. They applied the medial axis transformation and determined the head region, the body region, and the tail region using the geometrical properties of the rats.

Unsupervised learning methods have gained popularity in addressing the need for labeled data, offering valuable insights in terms of methodology. A group of approaches is built on adversarial frameworks [97–100]. [97] combines two GANs to generate labeled data for vessel segmentation in an unsupervised manner. One generator generates fake frames using fractal images and mask frames (grayscale angiogram images that serve as background images without the targeted vessels). A discriminator is employed to ensure realistic augmentation by using true angiogram images, while a segmentation discriminator is utilized to achieve realistic segmentations. The main focus of [98] is to provide a foreground object segmentation using an unsupervised deep learning framework incorporating geometric constraints in a GAN, based on a set of shape priors. Shapes can be exploited also based on object *prototypes* as described in [101], but the same kind of ”objects”, such as two persons in occlusion, will be segmented with this method as a single region. To provide training data for foreground object segmentation a GAN-based background generation method is introduced in [100]. A semi-supervised approach is presented in [102]. Basic augmentation is applied in a student-teacher type network to perform pseudo-labeling based on a set of labeled data. In this end-to-end approach exploits that the teacher model can be updated by the

DOI:10.15774/PPKE.ITK.2023.008

student model, to improve the generation of pseudo-labels for the unlabeled data, which are used to train the object detection model. For videos not only frame-wise approaches can be considered. In [103] a network is introduced that is trained based on the extracted dense local features from an RGB image and the optical flow between subsequent frames. These are treated as two complex structured representations. Object segmentation is predicted based on propagated pixel-level information by utilizing a spatio-temporal matrix to capture appearance and motion cues. In [104] segmentation is achieved by learning to group together image parts based on motion. The training utilizes videos, but the inference is performed for single frames. A *bootstrapping* process is proposed to decouple the objects of correlated motion. Transformers can be considered alternatives to deep convolutional network-based approaches. [105] provides a method for self-supervised pretraining a standard Vision Transformer(ViT) model, based on a student-teacher framework, to learn class-specific labels. Given an unlabelled pool of videos [106] can automatically find relevant samples based on a query, and provide a transformer-based instance segmentation in a self-supervised manner. The listed deep learning approaches provide diverse methods to reduce human time and effort in segmentation tasks. Similar to my approach, some methods apply generative models and the generation of synthetic training data, tailored to the characteristics of the input data, to train deep networks. Other approaches propose custom architectures, use subsequent frames, or transfer the knowledge from a limited number of labeled data. Thus, a range of potential directions for automatic annotation can be considered, particularly for complex inputs.

3.2 Methods

The goal is high-quality instance tracking, i.e., the minimization of the number of erroneous switches. I overview the algorithm selection method. The individual computational steps are detailed in the main part of this section.

DOI:10.15774/PPKE.ITK.2023.008

3.2.1 Overview of the Algorithm Selection Procedure

The optimization of segmentation is the first step of our heuristics. It is followed by a propagation algorithm.

I have tested multiple edge detection algorithms: the Sobel [90], the Canny [91] methods, and the trained BIPED model of the DexiNed [93] network. I dropped Sobel due to its low performance. I evaluated Canny and the BIPED model using F-score, precision, and recall metrics [107]. Canny and the BIPED model produced similar results (see later). In turn, I decided on using the trainable method, i.e., DexiNed, as it had a greater potential.

I trained the Edge Detection in two ways: from scratch and with transfer learning starting from the BIPED model [93]. I used different ground truth edge thicknesses and the epoch number of the training procedure.

The selection was based on the performance of those frames where instances overlapped and the foreground mask was not connected. The model that had the lowest number of frames with a single connected foreground region was chosen. I refer to this model for the scratch and the transfer learning cases as SEPARATS and BIPED-TL, respectively.

I used the selected models as follows. I estimated the foreground mask and generated edge images. If it was a single connected region then the algorithm of Edge Completion was invoked. If the foreground mask had more than one regions then I combined it with the results of the *body parts* method [56] by means of higher-level knowledge, the number of bodies (see later). In the case of a single connected foreground region only the *body parts* method was used. The outcome is the *per-frame segmentation*. Per-frame segmentations were connected by a propagation algorithm [57].

Below, I elaborate on the sub-components of our heuristics that give rise to the final evaluation pipeline. The overview of the main steps is depicted in Figure 3.4.

DOI:10.15774/PPKE.ITK.2023.008

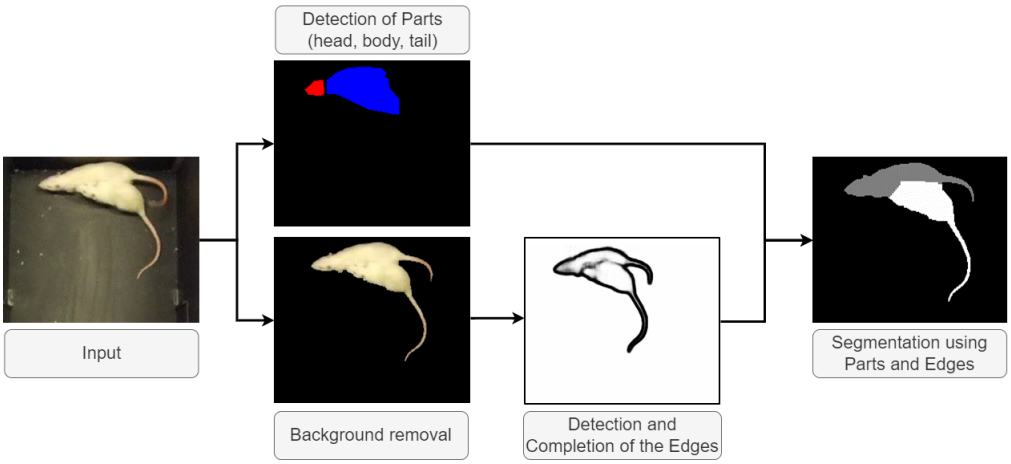

Figure 3.4: Sketch of the test pipeline for a single frame. Our aim is to maximize the segmentation precision within the frame to provide a strong basis for tracking. Body parts and edges are predicted and edge completion methods are invoked. Pre-processing provides the foreground masks and removes the background from the frame. A post-processing module combines the information and predicts the segmentation of the instances separately to enable identity tracking. Note the error in the last subfigure: the head of one of the animals is mislabeled. However, tracking remains error-free.

3.2.2 Pre-Processing

There are two animals in the scene. They are neither allowed to leave the cage nor able to hide, but while they are interacting, they can heavily occlude each other, making tracking challenging.

Thus, I differentiate between frames with disjoint and occluding object instances based on foreground segmentation. For the latter case, i.e., frames with a single connected foreground region, for a given instance segmentation method I will differentiate between frames with *separated* and with *unseparated occluding object instances*.

I apply the DexiNed model (*BIPED model*), which was pre-trained on the first version of the BIPED dataset, as this performed better than later models (downloaded from `https://github.com/xavysp/DexiNed` accessed on 21 March 2021). In what follows, I use interchangeably the following words: contour, edge, and boundary. These – in contrast to the geometric definitions – represent here the set of pixels marked as edges

DOI:10.15774/PPKE.ITK.2023.008

by DexiNed or other edge detection methods. I distinguish two types of boundaries: external boundary separates the background from the rest, i.e., from the occluding or non-occluding two animals. The inner boundary refers to the boundary within the foreground mask. This inner boundary may separate the animals, as seen later. During pre-processing, I use (a) the original frames, (b) the edge images predicted by BIPED model, and (c) the foreground masks as inputs.

As the background is static in our case, one can estimate the background by taking the mode of 2000 frames. Then, I can get the foreground via background subtraction. I also incorporate intensity histograms in the foreground estimation process to make it robust against slight illumination changes and reflections, the reflections on the cage parts, and the shiny urine the animals add to the scene occasionally. The drawback of this method is that parts of the tails might be lost. However, I can neglect the tails, as the rodent's tail movement is not considered as a factor for tracking the rats. For frames with separated object instances, the disjoint foreground masks are labeled with index values, which I define as ID values of the instances. The edges predicted by the BIPED model are outside the foreground mask. I define the edge masks for each frame as the foreground mask and the instance contours detected by the BIPED model. The outputs of pre-processing are the foreground masks, the masks of the instances, the masked RGB instances, and the edge masks.

3.2.3 Augmentation

The automatic generation of annotated images is inspired by the idea in [108]. The applied synthetic training frame generation is a modified approach compared to the traditional concept. Rather than artificially generating all pixel values along a given distribution, I assign RGB pixel values of rats in actual images to the areas of the synthetically positioned rat masks.

I used frames with occlusion-free samples for constructing overlapping instances. For the training of the edge detection network, I also set the generating parameters to allow some non-overlapping positions, for the sake of comprehensive learning and to avoid

DOI:10.15774/PPKE.ITK.2023.008

overfitting on the significantly overlapping samples. For each input frame, I generated five rotated samples evenly distributed along with adding a random correction. I added two translated copies to each of them. The amplitude of translation was set according to a heuristic using the resolution of the frame, and the sizes of the rats relative to the frame aspect ratio. I applied these transformations to the RGB instances (region defined by the foreground mask) and for the masked edges, to create augmented RGB (aRGB) images and augmented ground truth (aGT) respectively. Unfortunately, if one puts two correctly segmented single rats huddling or overlapping with each other for aRGB images, the shadows are different from the true occluding cases. Note that in 3D, the unoccluded instance is *above* the other one. This unoccluded instance will be called *the upper instance*. In particular, the shadow of the upper rat on the lower one is missing and it is the very relevant part, i.e., at the separating boundary. The foreground mask size for rats in RGB frames has a high impact. If it is small and excludes the shadows, then the shape is strongly damaged and becomes unrealistic. If it is large for a given instance, then the shadow at the silhouette introduces an artifact that will be picked up by the deep network giving rise to failures on the real frames. The compromise is to minimize the damage to the shape while eliminating the artifacts by removing the unrealistic shadow from the RGB image with *inpaint-based blur*, as follows:

I define the separating boundary as the 1-pixel wide contour section of the upper rat within the mask of the overlapping instances, excluding the joint external boundary pixels. I use the inpaint function with the Telea algorithm [109] of the OpenCV package. Inpainting is invoked for the separating boundary, with a window radius of 7 pixels, to create a blur.

There is another pitfall that one needs to overcome: towards the external boundary, the blur is biased by the background pixels. I mitigated this by using an auxiliary background, I change the background to a rat-like one. I crop a region from the rats in a hand-picked frame of side-to-side contact. The cropped rat texture is resized to exceed the joint size of the overlapping instances and forms the background for us. This single resized image used for the augmentation of all RGB frames minimizes unrealistic

DOI:10.15774/PPKE.ITK.2023.008

artifacts for our case. Note that this particular approach is exploited only during the inpainting of the augmented RGB frames. Otherwise, the background is set to zero.

In aGT edge images only the edge pixels of the rats are non-zero. This is to minimize the influence of background prediction on the loss. To reduce the large number of background pixels, the area of the augmentations is set to a small, but sufficiently large square to enclose the overlapping instances. In our experiments, I used a square of 256×256 pixels. The generated synthetic dataset of augmented inputs and aGT edge images are used in the training of all three networks presented in Sections 3.2.4 and 3.2.5.

3.2.4 Training Edge Detection

I trained the network architecture presented in [93] with our synthetic dataset, described in Section 3.2.3. I present the illustration of this step in the Training I column of Figure 3.5.

I used 92 330 input images with corresponding augmented ground truth edges. This trained model is the first deep network in the prediction pipeline. I trained multiple models for edge detection. I evaluated the models as described later, in Section 3.3.1 and picked the best in terms of separating the instances. This best model is called SEPARATS .

3.2.5 Training the Edge Completion

In frames with occluding instances the foreground mask, apart form the tails, is a *single connected region*. The inner boundary as defined before separates the two instances. In what follows I refer to this part of the edge structure as the separating edge. If the separating edge is continuous, the upper instance has a closed contour corresponding to its mask boundary. I call the frames with overlapping instances *frames with* unseparated occluding object instances if the inner edge is discontinuous. Hence, in this case, there are less than two disjoint regions within the foreground mask.

DOI:10.15774/PPKE.ITK.2023.008

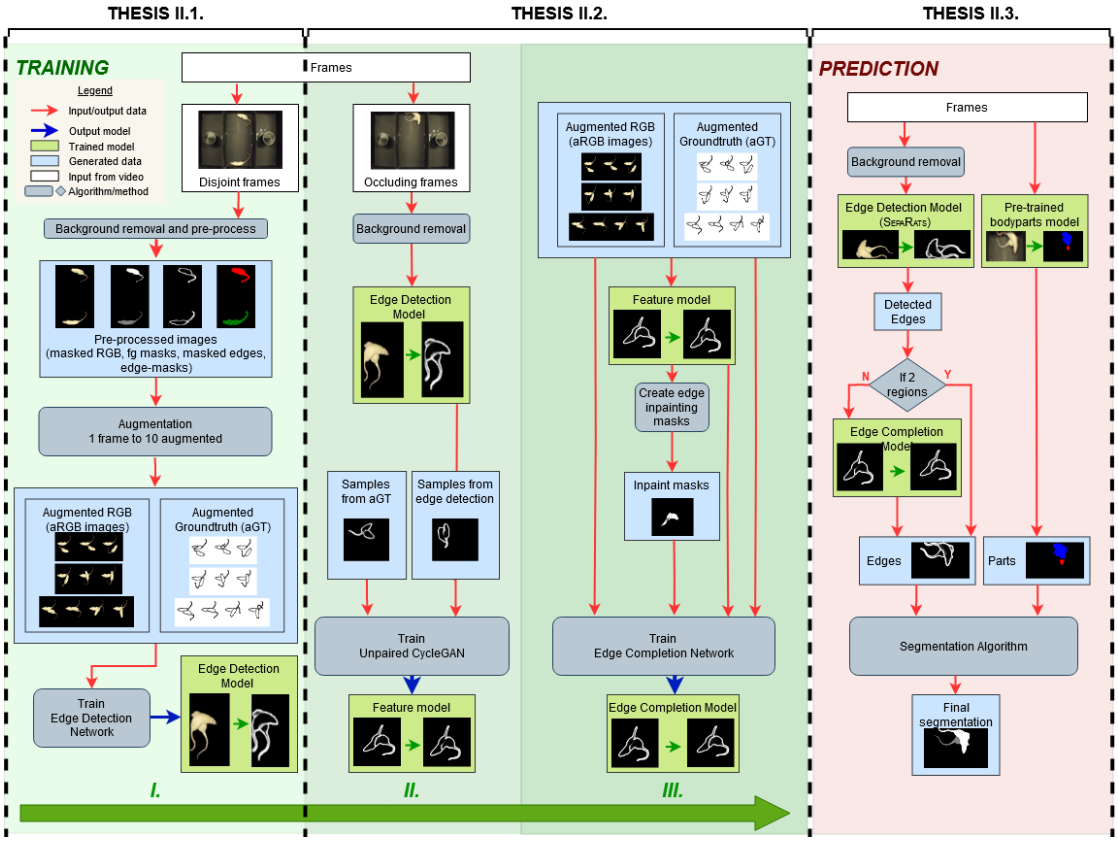

Figure 3.5: Illustration of the training and prediction pipelines. Three main blocks of the training pipeline: I. synthetic data generation and training Edge Detection model; II. training the Feature Model; III. extending synthetic dataset and training the Edge Completion model. Training is built upon synthetic data generation. Overlapping inputs and augmented ground truth data are constructed from pre-processed frames with non-occluding instances. The trained Edge Detection network is applied on frames with occluding instances. The unpaired CycleGAN [110] generates training data on the synthetic dataset for training the Edge Completion network. The prediction pipeline applies the Edge Detection model for the foreground of the frames. If the detected edges are not separating the instances, the foreground mask is a single connected region and the trained Edge Completion network "extends" the edges inside the foreground mask. The segmentation algorithm predicts the final segmentation for each frame. The edge regions and the body regions detected by a pre-trained model [56] are combined to provide a reliable segmentation for identity tracking of the highly similar and markerless instances including during heavy overlaps. The proposed pipeline is completely automatic, no human annotation is required. For more details, see the text.

DOI:10.15774/PPKE.ITK.2023.008

I train an inpainting network to connect the edge endings in imperfect edge images (i.e., with unseparated object instances). For this step, I extend the augmented data (aGT and aRGB images) with corresponding imperfect edge images by transforming the continuous separating edges in the aGT edge images to non-continuous sections.

It is hard to formalize and mimic the characteristics of edge endings of missing contour parts for Edge Detection prediction with basic image processing methods,therefore I use the unpaired CycleGAN proposed in [110] instead. To train a CyleGAN model, I apply the trained SEPARATS Edge Detection model to frames with occluding instances of the training set. I randomly select 2 200 predictions where the instances are unseparated, and an additional 2 200 aGT edge images. I generate edge images with discontinuous separating edges using the trained CycleGAN model (denoted as Feature Model in Figure 3.5) for each aGT edge image, which provides input and desired output pairs for training an edge inpainting network, the Edge Completion Model illustrated in Figure 3.5, column Training II.

To train the edge completion network I use the same synthetic dataset as in the training of the Edge Detection network but extended with the imperfect edge images. I build our method on the generative adversarial network (GAN) proposed in [95]. This network uses three inputs: an RGB image, an inpaint mask that defines the area of missing edges, and the edge map to be completed. As opposed to [95], I do not apply masks to the RGB input of the generator network, because the aRGB image is always fully observable. I generate the inpaint mask in a way to ensure that no information is lost: it must cover the complete region of missing edges and at the same time exclude known boundaries: both the external contours and the inner boundaries predicted by the CycleGAN Feature Model. I determine the mask of boundary pixels $\mathcal{B}$ by applying a threshold to the boundary image. Thus, the formula for calculating the inpaint mask $\mathcal{M}$ is as follows:

$$\mathcal{M} = \mathcal{F} \setminus \mathcal{B}$$

where $\mathcal{F}$ denotes the foreground mask, and the operator $\setminus$ stands for set difference.

DOI:10.15774/PPKE.ITK.2023.008

I note that in the prediction the inpaint mask can be generated in an analogous way, with the difference that the boundary image is provided by the Edge Detection model.

Because the primary goal of the edge completion model is to make the separating edge continuous for overlapping instances, this region needs to have a higher weight in the loss function than the non-boundary regions of the inpainting mask. Thus, I define the loss $\mathcal{L}$ of the discriminator for each frame as

$$\mathcal{L} = BCELoss(\mathcal{M} \circ E_{pred}, \mathcal{M} \circ E_{aGT}) + BCELoss(\mathcal{S} \circ E_{pred}, \mathcal{S} \circ E_{aGT})$$

where *BCELoss* stands for adversarial loss with Binary Cross Entropy [111, 112] objective, and the operator $\circ$ denotes the Hadamard product [113], E_{pred} and E_{aGT} are the predicted edge image of the inpainting generator and the corresponding aGT edge image, respectively, and $\mathcal{M}$ denotes the inpaint mask. $\mathcal{S}$ is the mask of the separating edge. This mask of the separating edge consists of the missing part of the edge that I want to complete and is dilated to include a few pixel environment next to the edge. The Hadamard product restricts E_{pred} and E_{aGT} to the mask region while keeping the grey values [95].

For the discriminator, I modified the input with a re-scaling on the output of the generator as follows. The separating edges belong to the upper rat's mask by construction, so I apply a heuristic: those edges that belong to the upper rat were given a weight of 1, whereas those that belong to the lower (occluded) rat, i.e., boundaries not related to the upper rat and thus containing only external boundary pixels of the overlapping instances are given a weight of 0.5. This weighting emphasizes the upper rat's boundary in the discrimination procedure being critical for the closedness of this contour. I illustrate the training of the Edge Completion Model in column III of Figure 3.5.

DOI:10.15774/PPKE.ITK.2023.008

3.2.6 Segmentation

See pseudocode for segmentation in Algorithm 1. The segmentation is the last step of the first stage of processing single frames. It is the post-processing of the information from body parts and edges. I focus on the occlusion of bodies and remove tails with a binary opening from the foreground mask of the input frame. The tails are not useful for the segmentation of the bodies and even represent a noise due to the various possible shapes.

Algorithm 1: Segmentation

Input: Grayscale edge image ($G \in [0,255]^{H \times W}$), foreground mask ($M \in \{0,1\}^{H \times W}$), array of predicted body region masks ($B = \{B_1, B_2\}$, where $B_j \in \{0,1\}^{H \times W}$ $(j = 1,2)$)

Output: Segmented image with labels ($I \in \{0, ID_1, ID_2\}^{H \times W}$)

Find edge regions (L and R) using Algorithm 2.
if $L = 1$ **then**
    Split edge region R_1 into multiple regions based on the predicted body masks B.
if $L \geq 2$ **then**
    Assign edge regions to the most appropriate bodies based on overlap (A) using Algorithm 3.
    Get the final instances (I) from the assigned edge regions A by applying watershed, bounded by the foreground mask M.
else
    In the final segmentation I fill each pixel of the foreground mask M with a flag value denoting unseparated instances.

The combination of the information from edge and body parts detection consists of two main subroutines, as follows. First, I determine the regions corresponding to the detected edges. I use thresholding and binary morphological operations to transform predicted grayscale edges into a binary edge image. I used a randomly chosen subset

DOI:10.15774/PPKE.ITK.2023.008

of the training samples to select the threshold value. The output of the edge detection network is an intensity map, with higher intensities denoting higher certainty of the presence of an edge. The algorithm has two types of pitfalls (limiting factors). For high threshold values weak edges, such as the edge between the instances, get eliminated. In this case, the foreground is made of a single connected region. For low threshold values, two or more regions may appear. In addition, edge regions can cover considerable areas including a large part of the foreground, giving rise to high noise for the analysis following this thresholding. Upon visual inspection and considering the two limiting factors I set the threshold value θ to approximately 25% of the maximum ($\theta = 60$ in the [0, 255] range) and set pixel values above (below) this threshold to 1 (0). This value is somewhat arbitrary and could be optimized if needed. I apply further transformation for thinning, and to reduce the noise introduced by this limit value, as follows. I call the resulting non-zero regions within the binary edge image *edge stripes*. The edge stripes are assumed to cover the contours and they can be up to 10 pixels wide depending on the output of Edge Complete and the parameters of the algorithm. To thin the stripes, I apply medial axis skeletonization of the *scikit-image* package [114] and calculate a midline (thresholded skeleton) of the edge stripes. The midline approximates the contour of the instance shapes. I add a dilation step for eliminating potential pixel errors and have a single connected region that splits the foreground mask into regions. Each region is labeled uniquely.

I provide the pseudocode in Algorithm 2.

Algorithm 2: Find edge regions

Input: Grayscale edges ($G \in [0,255]^{H\times W}$)
Output: Edge-based labeling for regions ($R_i \in \{0,1\}^{H\times W}$) ($i = 1,2,\ldots,L$), number of edge regions (L)

1 Threshold the grayscale edges G with $\theta = 60$
2 Apply morphological closing to get binary edges (B).
3 Apply medial axis transformation to the binary edges B to get its midline (M).
4 Find distinct regions (R) (L pieces labeled with $1,\ldots,L$) bounded by the midline M.

DOI:10.15774/PPKE.ITK.2023.008

Second, I combine this information with the predicted body regions from parts detection, using the method proposed in [56]. In Figure 3.6 I illustrated the components of the creation of initial labeling for a frame.

(a) (b) (c) (d)

Figure 3.6: Initial labeling. (**a**) Grayscale input image (**b**) Dilated midline of the detected edges (**c**) Body parts prediction with initial labels shown by different colors (**d**) initial labels from (**b**). (**c**,**d**) are used for further segmentation steps in Algorithm 3.

This approach is a Mask R-CNN-based [84] method, trained on synthetic data, without any human annotation. From the detected labeled regions (head, body, and tail), I use the predicted body regions in our combined method, while ignoring any heads and tails. Bodies may overlap or have irregularities in the shapes due to outlier contour points. This would bias our method; therefore, a correction is needed, as follows. I filter this type of noise by formalizing the knowledge of the shape of the instances. I calculate the geometric center (centroid) for both bodies by taking the mean for the row and column coordinates. Pixels from the intersection area of the bodies are assigned to the closest center of gravity. See Figure 3.7 for our illustration.

DOI:10.15774/PPKE.ITK.2023.008

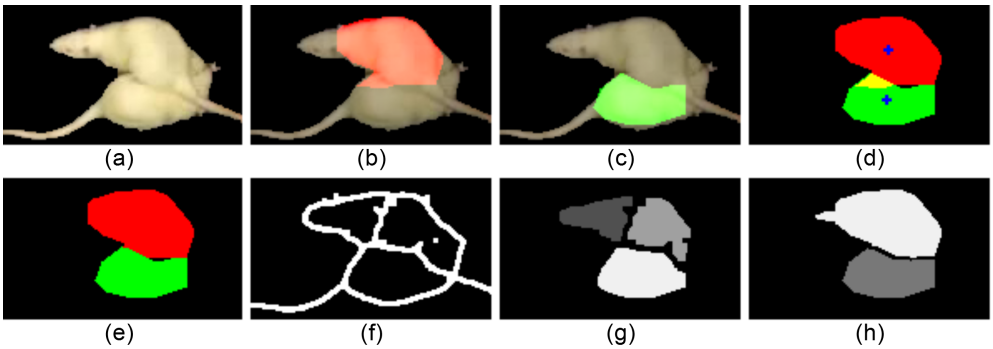

Figure 3.7: Due to identical appearances without any markers, region-based segmentation is often erroneous. If two bodies are predicted (shown in red and green), they are often overlapping, with non-realistic outlier contour points. I apply a centroid based method as a correction. (**a**) input; (**b**) first predicted body region; (**c**) second predicted body region; (**d**) both predicted body regions, yellow marks overlap, blue marks centroids; (**e**) after correction (each point in the intersection is assigned to the closest center of gravity); (**f**) edge midline Algorithm 2; (**g**) initial region labeling based on (**f**) and dilation; (**h**) region labeling combines (**e**,**g**) using Algorithm 3.

For cases when rats heavily occlude each other, the separation of the bodies is hard, often a single joint body region is predicted only. In Figure 3.6 the main components are illustrated for calculating the labeled image for a frame. I assign predicted body regions and edge regions to each other, based on the ratio of overlap. If a single predicted body region is assigned to all edge regions, then body information is ignored, and initial labeling is created using only in the edge regions.

If two predicted body regions are detected, each edge region is unioned with the belonging body mask, and unique labels are set for united regions. Details can be found in Algorithm 3. I have no information about instance identities within a single frame; therefore, for simplicity, I label the bigger region with label 1, the smaller one with label 2. For any pixel which would have both labels, I set the label of the background, $L_{bg} = 0$. These labels form the initial markers for a watershed algorithm [114, 115] to fill the foreground mask and return the instance segmentation.

DOI:10.15774/PPKE.ITK.2023.008

Algorithm 3: Assign edge regions to bodies

Input: Edge regions ($R_i \in \{0,1\}^{H \times W}$ $(i = 1, 2, \ldots, L)$), array of predicted body region masks ($B = \{B_1, B_2\}$, where $B_j \in \{0,1\}^{H \times W}$ $(j = 1, 2)$)

Output: Initial labeling for regions ($A = \{A_1, A_2\}$, where $A_j \in \{0,1\}^{H \times W}$ $(j = 1, 2)$)

To get the initial labeling (A_1), assign the edge regions R_i to B_1 based on the overlap: $A_1 = \bigcup \left\{ R_i | 0.5 \leq \frac{R_i \cap B_1}{R_i \cup B_1} \right\}$.

if *all edge regions were assigned* **then**

    Ignore body masks, and use edge regions R_1 and R_2 as initial labeling A_1 and A_2, respectively.

else

    Assign the remaining edge regions R_i to B_2: $A_2 = \bigcup \left\{ R_i | \frac{R_i \cap A_1}{R_i \cup A_1} < 0.5 \right\}$.

    Improve the assigned regions A based on the body masks B.

If the segmentation map contains only a single instance ID then the pixels of the foreground region are marked by a flag indicating that Algorithm 2 was not able to separate the instances. This flag may be used for further processing either using information propagation methods, such as in [57] or to call a human annotator to examine and correct.

3.2.7 Frame Sequence Propagation

Our main concern is to avoid ID switches during tracking. To improve the prediction for a single frame, I use the information from neighboring frames. We used the propagation approach presented in RATS [57], with slight modifications. Due to the different characteristics in appearance between the predictions of the Mask R-CNN model used by RATS and our proposed instance segmentation method, the parameters had to be tuned. We also incorporated optical flow (OF) [116] in the propagation to improve its reliability. We propagated the frames which were preserved by the rule-based method of [57], by shifting the pixels with the corresponding OF. However,

DOI:10.15774/PPKE.ITK.2023.008

this works well for short sequences, but the successive application of pixel-wise OF transformation can smear instances masks for longer timeframes. For this reason, we only used the propagated masks to guide the bipartite matching process [117] and used the original predictions for the final output.

3.3 Results and Discussion

I used 15 400 frames for training from a sample video. These frames correspond to about an 11-min video, which is very short when compared to the length of the necessary recordings needed for pharmaceutical studies[1].

I used a hand-annotated representative subset of 3 600 frames from the surveillance video recorded during biological research aiming at measuring animal behavior under medical treatments. This is the ground truth data for the evaluation. These images were not used for training. The ratio of images with heavily occluding instances is around 15%, and the distribution of these subsequences in the complete sequence is nearly uniform. My goal was to minimize human annotation and attention time, therefore, time measurements are only indicative. 15 epochs for DexiNed took 6 hours, 10 epochs for CycleGAN took 1 hour, and 20 epochs for EdgeConnect took 16 hours[2]. Edge Detection model generated predictions with a speed of 6.66 frames per second, Edge Completion with a speed of 1.66 frames per second. Both measurements include file operations, code optimization would improve running times.

For medical research, it is crucial to reliably maintain the identity of the instances. Therefore, to evaluate the performance of the tracking, I use the track switches (TS) metric from [57], which measures the number of instance ID switches during tracking. When only trajectories were available, such as in the case of ToxTrac [78] and idtracker [76], the track switch evaluation was conducted by human observers. To evaluate segmentation quality I use Intersection over Union IoU, also known as the Jaccard index [118]. This

[1]The number of images used for training and testing each deep network of the pipeline is shown in Table 4.1.

[2]On a single GPU of a server equipped with two Nvidia TITAN RTX GPUs @ 24 GB memory/GPU, AMD Ryzen TR2920X CPU @ 64 GB RAM

DOI:10.15774/PPKE.ITK.2023.008

metric gives us information about the amount of mislabeled pixels. For a prediction P and a ground truth G, IoU is calculated as $IoU = \frac{P \bigcap G}{P \bigcup G}$. For benchmark measures (F Mean, F Recall), I used the same ones as in [57], with the implementation taken from the Davis Challenge [119].

I compare my results to previously published literature methods, which are designed for tracking without human annotation: idtracker [76], idtracker.ai [77], and the ToxTrac [78] implementation of ToxID. An illustrative comparison is shown Figure 3.8.

As ToxTrac and idtracker do not provide per-frame segmentation masks, I relied on the GUIs and output videos by the authors and the output videos I generated. In the case of idtracker.ai, I used the codes published on the web (`https://gitlab.com/polavieja_lab/idtrackerai_notebooks/-/tree/master/` (accessed on 28 October 2021)) to export the segmentation masks and compared the pixels numerically to our automatically predicted segmentation. For all methods, I determined the number of identity switches using the videos. Results are presented in Table 3.1.

DOI:10.15774/PPKE.ITK.2023.008

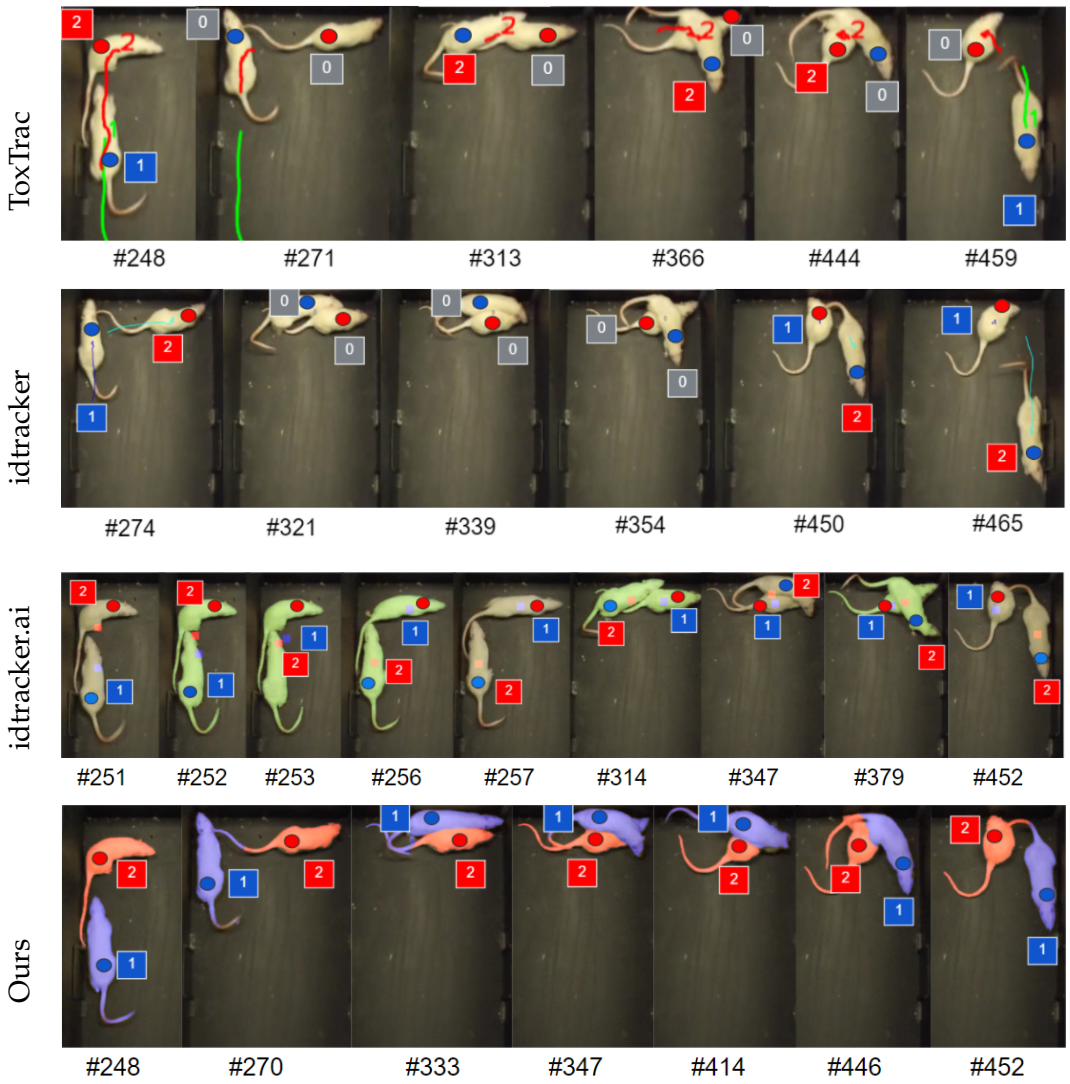

Figure 3.8: Illustrative visualization of methods compared. I use the track switches (TS) metric from [57], which measures the number of ID switches during tracking. Errors are shown for ToxTrac, idtracker, idtracker.ai in the first three rows. The results of our method (with SEPARATS) for the most challenging frames are given in the last row showing no errors. The small circles denote the ground truth IDs (blue for instance with ID 1 and red for instance with ID 2), the squares denote the predicted ID labels. Gray denotes ”lost ID”, missing ID predictions.

DOI:10.15774/PPKE.ITK.2023.008

Table 3.1: Comparison of segmentation-based trajectory tracking of methods which do not require prior data annotation, on our test data of 3 600 frames. TS denotes the track switches metric from [57], which measures the number of ID switches during tracking. Lost ID: there are less than two different foreground labels in the frame. For ToxTrac and idtracker, per frame segmentation masks are not available. Superscript 1 marks that the result is computed by means of the GUIs of the cited method, the best we could do for comparisons. BIPED-TL is the model trained with the augmented dataset with transfer learning on the original BIPED model [93]. Gray highlight: Mean of the Intersection over Union (IoU) values for all frames. For benchmark measures, we used the same ones as in [57].

ID Tracking Results							
Approach	**Num. of TS.**	**Num. of Frames with Lost IDs**	**IoU Mean**	**IoU & F Mean**	**IoU Recall**	**F Mean**	**F Recall**
ToxTrac [78]	9	267[1]	N/A	N/A	N/A	N/A	N/A
idtracker [76]	8	1055[1]	N/A	N/A	N/A	N/A	N/A
idtracker.ai [77]	10	1485	0.5556	0.604	0.59	0.652	0.746
BIPED & Parts	4	2	0.833	0.871	0.978	0.908	0.985
SEPARATS & Parts	0	0	0.846	0.883	0.994	0.921	1.000
BIPED-TL & Parts	0	0	0.845	0.883	0.994	0.921	0.999

DOI:10.15774/PPKE.ITK.2023.008

The mean *IoU* values are similar for the last three methods of Table 4.2 and are considerably higher than that of the idtracker.ai method, the third row in the table. The methods may predict a single connected foreground region (column *number of frames with lost IDs*). While the BIPED & Parts approach produces only two such frames, still the number of track switches is higher. This is due to discontinuities in the separating edge: BIPED can produce uneven (smaller and larger) regions and that noise can spoil the propagation algorithm. This demonstrates that track switches, the primary failure of ID tracking, may occur even with a relatively high *IoU*, 0.833 in this case.

For the sake of completeness, in the following subsections, I present the evaluations of the edge detection modules and the combined segmentation algorithm of our proposed pipeline.

3.3.1 Edge Detection and Completion

I trained the Edge Detection network with the augmented dataset with transfer learning on the original BIPED model, named BIPED-TL, and also from scratch, named SEPARATS. I trained these two models for 15 epochs on the 92 330 images using about 10% of the images for validation. The aGT method (described in Section 3.2.2) performed the best from a set of different augmentations that I created. For example, I used one pixel wide (aGT_{1px}) and two pixel wide (aGT_{2px}) contours of the foreground masks, but their performances were lower. For evaluating Edge Detection and Edge Completion I used the frames with occluding instances and neglected the easy non-occluding cases. In Figure 3.9 I show illustrative edge detection results for three frames with occluding instances.

The two most promising models (SEPARATS and BIPED-TL) are in a close range, which means that the model exploited the training data to learn the main characteristics of the input, which is significantly different from the original, BIPED dataset.

The task of identity tracking is highly tolerant to distortions and shifts of the estimated edges as opposed to the gaps in the edges. Traditional measures treat the precision of the pixels of the edges instead of their continuity. Continuity is most important in the inner

DOI:10.15774/PPKE.ITK.2023.008

part of the boundary for separation and tracking. Figure 3.10 illustrates the significance of continuity. Therefore, to compare edge detection methods in terms of the addressed task, I apply both direct and indirect evaluation approaches, similarly to [120].

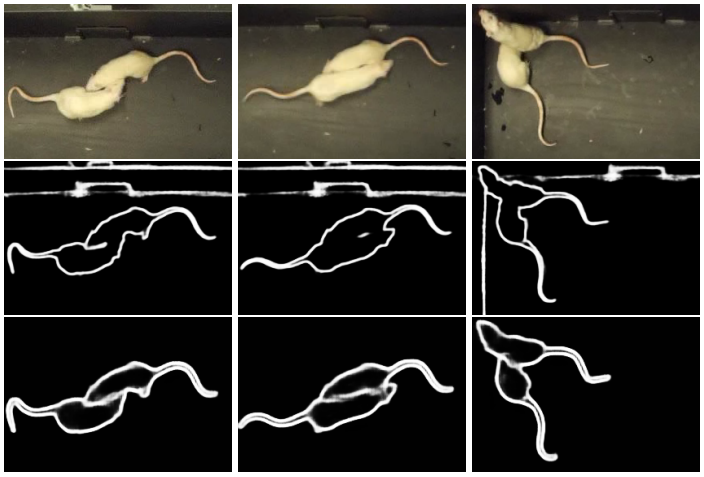

Figure 3.9: Edge detection results. (**Top row**): three different RGB frames. (**Middle row**): corresponding edge detection with BIPED DexiNed. (**Bottom row**): our Edge Detection results, with SEPARATS model. Our Edge Detection predicts well-defined grayscale edges in the inner sections of the boundaries that can be thresholded.

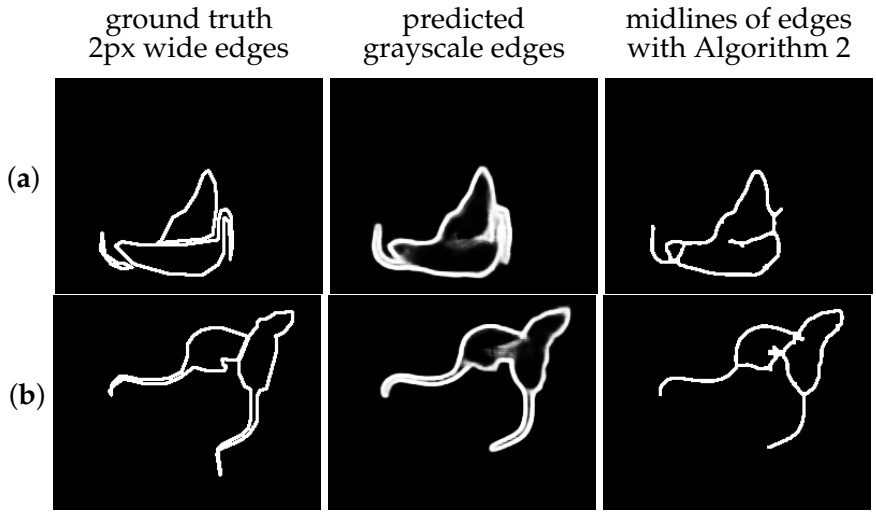

Figure 3.10: Illustration for the edge measurements. The F-score of the predicted grayscale edges in (**a**) (F = 0.6069) is better than the F-score of (**b**) (F = 0.5610). The predicted edges for (**b**) are distorted, but the extracted midline allows segmentation for both instances separately, as described in Algorithm 1.

DOI:10.15774/PPKE.ITK.2023.008

To select the most promising approaches, I consider the traditional measures (recall, precision, and F-score). F-score is calculated as $\frac{2PR}{P+R}$, where P stands for precision, R for recall between the ground truth and predicted contours. I show the results of the edge-related methods for F-score, precision, and recall [107] values using the implementation of the DAVIS challenge [118].

Edge error measures are instructive in selecting promising approaches. Both BIPED-TL and SEPARATS improved the edge prediction of inner edges, compared to the pre-trained BIPED model and the Canny algorithm. The results are presented in Table 3.2. F-score and recall increased to 0.4362 and 0.4781, respectively, using Edge Completion after SEPARATS.

Although an edge prediction that exactly matches the ground truth is a sufficient condition for instance segmentation, it is not definitely necessary. As opposed to a non-distorted separating edge that does not close the contour of the instance (with higher F, P, and R), a closed but distorted contour of the "upper rat" (with lower F, P, and R values) is sufficient for segmentation and reliable identity tracking. Therefore, as an indirect evaluation of the edge detection methods I rely on the results in Table 4.2 (the segmentation and tracking quality of the combined algorithm using the parts based regions and the regions from the edge detection models). The most important evaluation method is the number of track switches. The results are in line with the ones in Table 3.2.

Table 3.2: Evaluation of different edge detection methods on our test sequence of 3 600 frames, on the inner part of the overlapping objects. F-score is a support value for our decision, for more details, see text. Bold fonts indicate the results for the proposed models.

Edge Detection					
Approach	**Deep Learning**	**Epoch**	**Transfer Learning**	**F Mean**	**Recall Mean**
Canny	N	-	-	0.3614	0.3838
BIPED	Y	24	No	0.3510	0.3033
BIPED-TL	**Y**	**8**	**Y**	**0.4526**	**0.4902**
SEPARATS	**Y**	**4**	**N**	**0.4328**	**0.4676**

DOI:10.15774/PPKE.ITK.2023.008

In addition, I evaluated the segmentation quality for all three models. There is only a slight difference between the results of SEPARATS and BIPED-TL models when used for segmentation and tracking. This indicates that the training had sufficient data to achieve optimal weights not only by transfer learning from the BIPED model but also from scratch.

I calculated the mean values for the 471 frames with occluding instances. When evaluating per-frame segmentation quality, the IDs of the segments are not yet known: the ID assignment happens in a later stage of the processing pipeline. The segmentation step only considers information from a given frame. Thus, for the case of two instances, I can compare them to the ground truth in different ways and compute the corresponding *IoU* values. I take the maximum of these values.

If only tails occlude, the method presented in Algorithm 1 provides a reliable segmentation since the bodies have disjoint contours. Therefore I evaluate *IoU* values for the complete instance masks and also with leaving the tails using the body part detection method, if available. I present the calculated recall, precision, and F-score in Table 3.3.

The BIPED model requires the complete input frame, not only the foreground. In the prediction, the background elements and noises are significantly present. To compare the methods I apply the dilated foreground mask on the BIPED edge images . Results in Table 3.3 show that for critical frames with overlapping instances trained models perform better.

3.3.2 Evaluation on unlabeled data

In addition to the measurements presented in 3.3 above on the 3 600, human-annotated frames, I evaluated the proposed pipeline on four videos, each half-hour long, adding up to a total of 224 577 frames, with similar animals and the same type of setup. However, a project currently in development aims to provide an automatic behavior annotation that includes these videos in the training set, but there is no pixel-wise annotation for these frames, only individual behavioral pattern

DOI:10.15774/PPKE.ITK.2023.008

Table 3.3: Evaluation of the segmentation method, combining edge-based regions (from the trained models SEPARATS and BIPED-TL, and the baseline BIPED) and regions from body part detection. The 471 frames of the test dataset with occluding instances were used for evaluation. *IoU* is calculated without tails. Mean values over the instances are shown for each metric. For all 471 frames, there are at least two connected foreground regions with different labels, i.e., both ID labels are present on each prediction.

	Segmentation			
Edge Detection Model	**IoU Mean**	**Recall Mean**	**Precision Mean**	**F Mean**
SEPARATS	0.8087	0.8870	0.9025	0.8888
BIPED-TL	0.7872	0.8779	0.8883	0.8710
BIPED	0.7455	0.8320	0.8359	0.8247

annotations are available for every sixth frame. There are no instance masks annotated on these videos, thus they can be considered unlabeled data for segmentation purposes, implying that segmentation quality measurements can not be performed. In the lack of applying the pixel precision metrics to the segmented instance masks, I measured the consistency of ID labeling and tracking by human observation. The light conditions of these four recordings are similar, but different from the lighting in the human-annotated dataset.Background estimation is performed every 2 000 frames to adapt foreground segmentation to the changing light conditions. Light also has a high impact on detecting the contour between occluding instances. Therefore, to make SEPARATS model more robust I extended the training dataset with augmentation by applying a set of ten different brightness-related color transformations (one being the identity operator), including gamma corrections, histogram equalization, sepia transform, and a white balance augmentation method [121] (see Fig. 3.11).

From every generated aRGB image the color augmentation creates three different samples for the training set, by choosing a random transformation from the transformation set.I extended the training set with the new augmented images, generated fully automatically, to adapt the edge detection model to the varying light characteristics in the unlabeled videos, thus improving the performance.The importance of lighting and shadows among the instances in close or occluding positions is well

DOI:10.15774/PPKE.ITK.2023.008

(a) aRGB (b) white balance for Cloudy, Camera Standard [121] (c) white balance for Shade, Adobe Standard [121] (d) applying a sepia image transformation kernel (e) histogram equalization

(f) gamma correction with 0.8 (g) gamma correction with 1.5 (h) contract correction (i) gamma correction of contrast corrected image with 0.8 (j) gamma correction of contrast corrected image with 1.5

Figure 3.11: Illustration for the ten brightness-related color augmentation methods. Each aRGB image is augmented with three different color transformations, chosen randomly with 10% probability for each.

reflected by the fact that the segmentation pipeline with the original SEPARATS edge detection model made 3-4 ID switches on average in the half-hour long videos because it failed to segment the instances on enough many frames so that tracking was not able to overcome them all. Using the model trained on the dataset with color augmentation, the masks overlayed on the videos demonstrated that the instance masks stay consistent with the instances, resulting in reliable tracking without any ID switches on any of the four videos. Therefore, the prediction pipeline with the trained models will be able to provide reliable input for the behavior analysis module.

3.3.3 Ablation Analysis

Key components of the segmentation compute (a) the edge-based regions and (b) the parts-based regions. I evaluated the combined segmentation method of Algorithm 1 with both components (edges and parts) in the prediction. For comparisons, I also

DOI:10.15774/PPKE.ITK.2023.008

evaluated the two components separately using all three edge models, i.e., SEPARATS and BIPED-TL , and BIPED as a baseline. Table 3.4 shows the Jaccard index (*IoU*) [118], the average values of recall, precision, and F-score for the predicted segmentation masks.

I evaluate per-frame segmentation in these ablation studies without using any information from the neighbouring frames and for those frames when the method could separate the instances. The number of frames with unseparated object instances was between 84 and 106 for the ablated methods with the models trained on synthetic rodent datasets (cca. 17–22% of occluding frames), but the frames were different for each method.

Since dropped frames are critical, the aim of this evaluation is to show the precision of the segmentation for the frames with successfully separated object instances. Results are shown in Table 3.4.

Table 3.4: Comparison of per-frame segmentation on the 471 frames with occluding instances. In lines (1), (2), and (3) I present the results for the combined segmentation method with both modules: Edge Detection and body parts regions. In (4), (5), (6), and (7) I show the quality of the segmentation for the modules separately. These values are marked by * to indicate that evaluation is only performed for the frames where two different foreground labels were predicted. Right-most column: number of dropped frames for the different methods.

	Segmentation				
Method	**IoU Mean**	**Recall Mean**	**Precision Mean**	**F Mean**	**No. unsep[3] frames**
(1) SEPARATS & parts	0.7991	0.8895	0.8897	0.8830	0
(2) BIPED-TL & parts	0.7872	0.8779	0.8883	0.8710	0
(3) BIPED & parts	0.7456	0.832	0.8359	0.8247	0
(4) parts	0.8295 *	0.9147 *	0.9008 *	0.9047 *	85
(5) SEPARATS	0.7603 *	0.8657 *	0.8689 *	0.8559 *	84
(6) BIPED-TL	0.7826 *	0.8725 *	0.8915 *	0.8694 *	106
(7) BIPED	0.7925 *	0.8653 *	0.8920 *	0.8722 *	384

We have seen in Section 3.3.1 that although SEPARATS is slightly weaker in terms of edge precision than BIPED-TL, it performs better in the segmentation quality of frames with successfully separated object instances.

[3]Number of frames with unseparated instances

DOI:10.15774/PPKE.ITK.2023.008

Chapter 4

Summary

4.1 Methods of Investigation

To address the question of adapting a Particle Filter to GPU I relied on the available literature on parallel, distributed and local particle filters [33–42, 45] during my research in 2013, considering the following major aspects. First, information share ratio among the particles to minimize degradation of the quality of estimation compared to the original algorithm, which resamples according to the complete cumulative distribution. Second, characteristics of the different GPU memory types, to improve kernel running times and on-device approaches for synchronization and random number generation to minimize the time consuming transfers between CPU and GPU. Although, NVIDIA Mersenne Twister included in the CUDA SDK seems to offer a promising solution, the distribution of the random numbers proved to be insufficient for low amount (hundreds and even thousands) of numbers. Therefore, I explored possible solutions and finally proposed two different approaches for random number generation.

To compare the quality of the estimation and the running time I used two benchmark models applied for evaluation in several state-of-the-art articles, available by the time of the research. On the one hand, the non-autonomous, non-linear model with a continuous state space used in [17, 68–70], and on the other hand, a bearings-only tracking model presented in [60] and used in [39, 40]. For evaluation, I used an NVIDIA GeForce GTX 550 Ti GPU with 1-GB GDDR memory, compute capability 2.1, and CUDA toolkit 4.1 with driver version 295.49.

To improve the quality of tracking highly similar occluding instances with keeping

DOI:10.15774/PPKE.ITK.2023.008

identity labels based on instance segmentation I considered the use of composite AI techniques as described in the Chapter 1. If different approaches as sub-modules are combined in a pipeline, the accuracy and performance can be better than that of an end-to-end approach [49, 50], moreover, each sub-module can be developed and optimized independently, and the results from one sub-module can be used to guide the development of the next sub-module. To improve instance segmentation the complete visual context (such as constraints) can be incorporated through higher logic into appropriately designed pipeline, while machine learning algorithms and computer vision techniques are used together. Combining two algorithms that can be considered as dual approaches [122], such as region detection and edge detection, allows for exploiting the complementary information that helps to mitigate the limitations of individual methods.

Based on the available literature deep learning methods are more promising for complex scenes than traditional edge detection techniques. Therefore I relied on the state-of-the art deep learning approaches presented in the literature [93–95, 120] to train an edge detection deep pipeline. To train the networks I considered the prediction quality and the time required for annotating the training data. To avoid the monotone and time consuming human annotation, I considered the method introduced in [57] and explored the required transformations to provide a similar characteristic to real, non-augmented test frames. The dataset is derived from a 20-minute, 25 fps observation video with a resolution of 1280x720, which was provided by the Department of Physiology and Neurobiology of the Eötvös Loránd University (ELTE). The frames used as image inputs are the same as those in the article [57], which were extracted and cropped to the region of the box, resulting in a final image resolution of 640x420 saved in *png* format. I synthetically constructed the training data from the first 10 minutes of the video containing 15 000 frames, as described in Thesis I.1 and Thesis 1.2. For training the Edge Detection [93] and Edge Completion [95] networks I used all 9 233 frames with non-occluding instances by determining the number of foreground objects based on histogram intensity and a background image constructed as a mode of 2 000 frames. For

DOI:10.15774/PPKE.ITK.2023.008

training the CycleGAN I selected 2 200 frames randomly from the ones with occluding instances. I chose the number of the training images for CycleGAN to be of a similar order of magnitude as the training dataset shown in the CycleGAN work, and based on the results, I considered it to be indeed appropriate.

To evaluate the quality of the segmentation results of the pipeline, 18 sequences of 200 images were selected by observation with challenging occlusions for human annotation. In this set 1 669 frames contain non-separated instances. See the number of training and test images for the different deep networks of the pipeline in Table 4.1. My goal was to minimize human annotation and attention time, therefore, time measurements are only indicative. 15 epochs for DexiNed took 6 hours, 10 epochs for CycleGAN took 1 hour, 20 epochs for EdgeConnect took 16 hours[1].

Table 4.1: Number of images for training and testing the different deep networks of the pipeline. 90% of the training images were used as an actual training set, and 10% as a validation set.

	Number of images			
	Train			Test
	RGB/aRGB	aGT(edge)	Non-closed contour	
DexiNed	92 330	92 330		3 600
CycleGAN		2 200	2 200	200
EdgeConnect	92 330	92 330	92 3300	3 600

The annotation was split between four annotators and each annotator's segmentation was validated by another annotator.

I extended the proposed pipeline with the tracking approach presented in [56] and compared it to state-of-the-art tracking methods [76–78, 81] by evaluating them on the 3,600 hand-annotated frames of a rat surveillance video. Although the main objective was to eliminate track switches, I also evaluated the instance segmentation quality of the methods (where it was applicable). For evaluations, we used two servers [2] one with two

[1] On a single GPU of a server equipped with two Nvidia TITAN RTX GPUs @ 24 GB memory/GPU, AMD Ryzen TR2920X CPU @ 64 GB RAM

[2] *nipg8* and *nipg10* of Neural Information Processing Group, Department of Artificial Intelligence, Faculty of Informatics, Eötvös Loránd University

DOI:10.15774/PPKE.ITK.2023.008

NVIDIA TITAN RTX GPUs with 24 GB memory/GPU, AMD Ryzen TR2920X@3.5GHz 24 core CPU, and 64GB RAM, and the other having two NVIDIA GeForce RTX 3090 GPUs with 24 GB memory/GPU, AMD Ryzen TR1920X@3.5GHz 24 core CPU, and 64GB RAM. Both servers had Ubuntu 18.04 operating system, codes were run in an Apptainer [123] virtual environment using Pytorch 1.4 and 1.9 for Edge Detection and Edge Completion networks respectively, with OpenCV 4.1.1. (see requirements and environment script file in the attached code).

4.2 New Scientific Results

THESIS I. A mapping of particle filtering onto the GPU architecture that can preserve local connections to prevent information loss

Particle filters can be considered as an extension of the Kalman filter, and therefore represent an attractive solution for Hidden Markov Model based problems in several fields, including image processing, robotics, and stock market forecasts. For computationally demanding approaches or applications required to run real time the GPU architecture allows efficient implementations. However, the resampling step of the original particle filter algorithm was considered unsuitable for parallelization without information loss, meaning a considerable limitation for speed-up. The algorithm presented in [45] exploits local connectivity of FPGA or Cellular Neural Network(CNN) chips. While other parallelized methods and distributed particle filters achieve high speeds with reduced information sharing between particles at the expense of accuracy, the algorithm presented in [45] is capable of better accuracy than the usual particle filter while drastically reducing the runtime.

In contrast to CNN and FPGA architectures GPUs, which are highly data-parallel, are widely spread devices that enable efficient computations. There have been some former implementations to GPUs, but the speed-up is highly limited at a cost of information loss in the resampling step.

DOI:10.15774/PPKE.ITK.2023.008

THESIS I.1. I designed an algorithm to map the Cellular Particle Filter (CPF) to the GPU architecture, in a way to exploit the GPU memory architecture efficiently to maximize speed, and at the same time maintaining the local connection property of the original CPF topology.

I developed a method, based on the cellular particle filtering structure, to provide information sharing, for GPU architecture. Shared memory with coalesced threads and synchronization on the global memory provides a faster computation than surface or texture memory and is not limited to 2D layouts. Therefore, I developed a method to map the local connectivity of a two-dimensional CNN architecture efficiently to the one-dimensional, read/write, fast-access on-chip memory of the GPU architecture (see Figure 4.1 for illustration of concept).

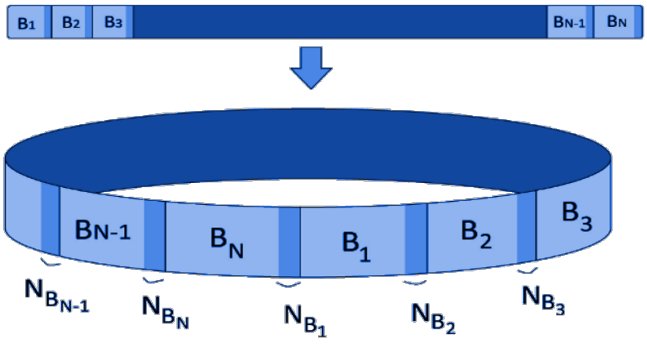

Figure 4.1: Restructuring linear representation of N blocks to a ring type topology. B_i stands for the i^{th} block, and N_{B_i} for the corresponding neighbourhood from the previous block, $i \in 1, \ldots, N$.

I showed by experimental evidence that the position error is similar to existing implementations on GPU, while the speed is better, although the possibility of one-to-one comparisons is limited by the variety of GPU devices used for measurements. Also, state-of-the-art works only show kernel runtimes. Those that use a device with similar performance to ours or provide no details about the used device are in the same range as our 77 ms total runtime[3], which includes I/O operations. Compared to GPU-implemented distributed particle filters, our algorithm preserves the local connectivity

[3]Measurements were done on a NVIDIA GeForce GTX 550 Ti GPU with CUDA toolkit 4.1, available at the time of the research in 2013

DOI:10.15774/PPKE.ITK.2023.008

of the particles, therefore it achieves the accuracy of the original filter, however with a total running time of less than 12 milliseconds at 16 thousand particles per state, which corresponds to a 164x speed-up compared to CPU implementation[4]. Meanwhile, the method of mapping the 2D layout of the processors and the preservation of the local connections to the 1D memory architecture allows other algorithms based on two-dimensional connections to be efficiently mapped to GPU.

Related publications of the Author: [A1, A3]

THESIS II. Unsupervised segmentation of highly similar occluding rat instances built on a pipeline of deep networks exploiting edge information

In terms of overall pipeline efficiency not only the implementation of each algorithm should be considered, but also the amount of human attention and time required. Identity tracking and instance segmentation are crucial in several areas of biological research. Behavior analysis of individuals in groups of similar animals is a task that emerges frequently in agriculture, pharmaceutical studies or behavioral ecology, among others, and usually requires a decent amount of human annotation. Automated annotation of many hours of surveillance videos can facilitate a large number of biological experiments, which otherwise would not be feasible. Solutions based on machine learning generally perform well in tracking and instance segmentation; however, in the case of identical, unmarked instances (e.g., white rats or mice), even state-of-the-art approaches can frequently fail, as shown in the number of track switches listed in the second column of Table 4.2. The challenging task of segmenting highly similar adjacent image regions can be addressed by traditional methods [A4, A5], but deep-learning-based methods offer a more promising direction.

We focus on data where mice/rats are very similar without any markers but may have received different medical treatments, and individual behavior patterns should be analyzed. In typical setups, the camera is fixed, and the foreground segmentation is

[4]The following NVCC compiler options were used to drive the GPU binary code generation, as proposed in the CUDA SDK Guide: -arch=sm_20; -use_fast_math. We also made some measurements with -arch=sm_13. The host c code was compiled with GCC 4.5; the compiler optimization flag was -O2.

DOI:10.15774/PPKE.ITK.2023.008

feasible, which simplifies the segmentation process. However, handling the changes in shape configurations and the heavy occlusions between the instances poses a significant challenge.

My approach is to build a pipeline inspired by *Composite AI* [51, 52], to provide reliable segmentation for identity tracking. Composite AI combines different learning architectures to achieve superior results and exploits human knowledge to improve overall performance.

The required time for tracking similar instances with human observation for each and every frame is highly time-consuming. In comparison, a partially automated solution can provide a significant speed-up. However, the time needed for the preliminary annotation for training and for the supervision in the prediction stage may still be considerable for each setup. I created a pipeline with an automatic annotation method to reduce the human need in the process. My method is illustrated in 4.2.

Evaluations show that a tracking method built on the segmentation using the predictions of the trained models further reduces the required supervision during the prediction stage compared to state-of-the-art methods.

THESIS II.1. I developed a method to generate a synthetic dataset fully automatically without any human annotation, based on only frames with non-occluding instances, to train an edge detection network for the detection of the separating boundary between highly similar occluding rat instances with a static background.

For the 3600-frame demonstration footage of the two rats, the instance annotation for tracking is a low-skilled task for a human annotator, taking twice the time of the video. For hours of surveillance videos, this is a significant amount of time that an expert must invest before performing the tasks that require expertise (analysis and inference). Moreover, this speed is achieved with the trade-off of using only one in six images, meaning that the temporal resolution, thus overall tracking quality, is degraded compared to a framewise method.

DOI:10.15774/PPKE.ITK.2023.008

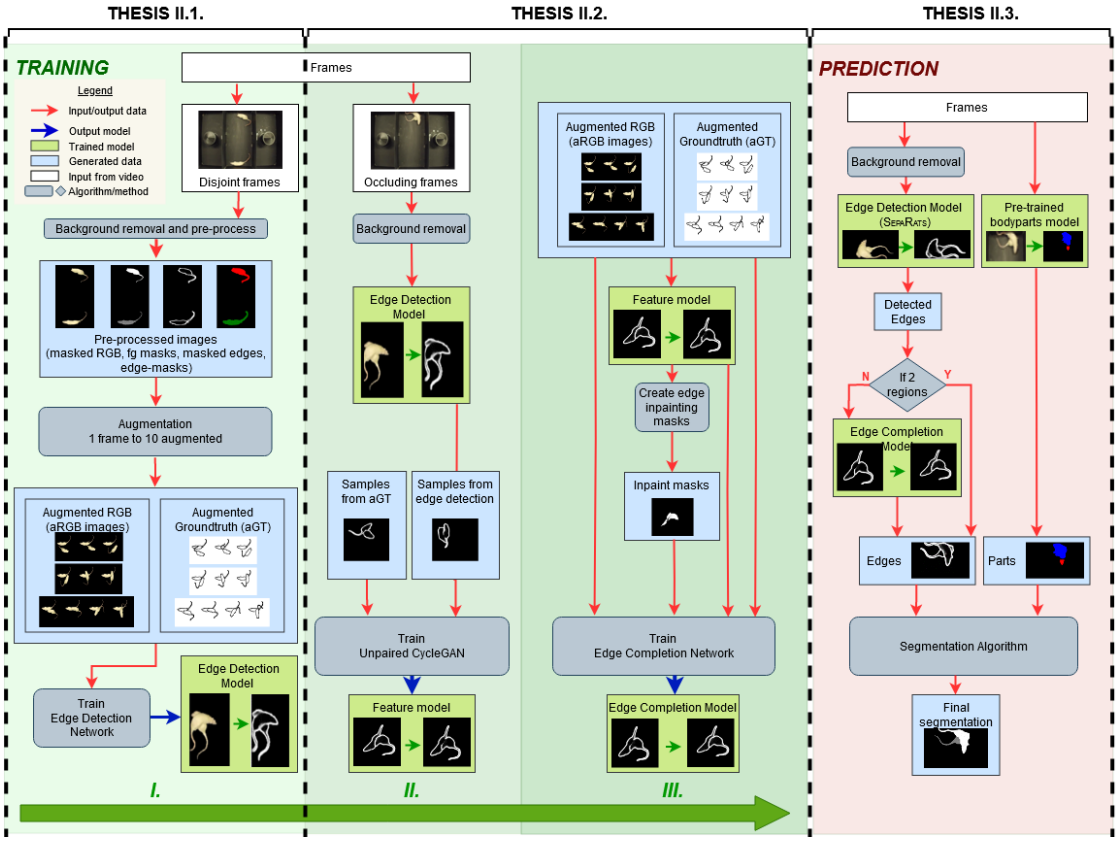

Figure 4.2: Illustration of the training and prediction pipelines. Three main blocks of the training pipeline: I. synthetic data generation and training Edge Detection model; II. training the Feature Model; III. extending synthetic dataset and training the Edge Completion model. Training is built upon synthetic data generation. Overlapping inputs and augmented ground truth data are constructed from pre-processed frames with non-occluding instances. The trained Edge Detection network is applied on frames with occluding instances. The unpaired CycleGAN [110] generates training data on the synthetic dataset for training the Edge Completion network. The prediction pipeline applies the Edge Detection model for the foreground of the frames. If the detected edges are not separating the instances, the foreground mask is a single connected region and the trained Edge Completion network "extends" the edges inside the foreground mask. The segmentation algorithm predicts the final segmentation for each frame. The edge regions and the body regions detected by a pre-trained model [56] are combined to provide a reliable segmentation for identity tracking of the highly similar and markerless instances including during heavy overlaps. The proposed pipeline is completely automatic, no human annotation is required. For more details, see the text.

DOI:10.15774/PPKE.ITK.2023.008

Manual annotation is hardly flexible to changes in the annotation protocol and thus can be revisited when new sub-tasks or changes in recording settings occur to ensure prediction quality. A trained deep network can significantly speed up processing videos if it provides a reliable automatic instance segmentation. We may synthetically generate training data based on instances that can be segmented with high confidence in some of the frames automatically, i.e. with foreground segmentation. However, it is essential that the synthetic data created by the augmentation method is sufficiently similar to real frames for the deep network. The masks of the objects preserve the inner details, but the contours of the generated objects also have to be realistic, both considering the background and the case of overlapping instances.

For very similar objects to create realistic occluding instances as an RGB training input, it is a challenge to accurately define the object masks in frames with not-occluding instances without losing important details, while noise from background pixels or shadows should not be added. Based on the background segmentation and that the number of instances is known and constant throughout the video my algorithm selects the frames with non-occluding foreground objects automatically. The edge images are also masked using foreground segmentation. An edge detection network [93] is trained with constructed data only. For each real RGB input of not-occluding instances, I generated 10 different overlapping positions, with slightly randomized parameters. For each position, I created both the augmented RGB (aRGB) and the augmented ground truth edge image (aGT). To overcome the noise in the aRGB images on the inner contours of the overlapping instances I applied an inpaint-based blur along the contour of the upper object. To further improve training quality I removed the background from both the aRGB and the aGT image.

With the trained model, within the frames with overlapping instances, the number of frames where the contour of the upper instance was not closed was reduced from 81.53% to 17.83%. This corresponds to a 4.57-fold improvement compared to the pre-trained state-of-the-art edge detection model (which is better than the traditional Canny [91], Sobel [90] methods).

DOI:10.15774/PPKE.ITK.2023.008

THESIS II.2. I designed a method with deep generative networks in a self-supervised approach to improve the separating contours in an edge image of a frame with overlapping objects.

I aim to further decrease the number of frames with occluding instances in which the contour of the upper instance is not closed after applying the model of Thesis I.1. and the missing part cannot be closed by applying binary morphological operators. I chose a deep generative network for edge inpainting. The required training data was generated without human annotation to maintain the self-superwised pipeline. Note that in 3D, the unoccluded instance is above the other one. This unoccluded instance will be called the upper instance.

I use a Generative Adversarial Edge Completion Network, based on [95] for the frames where the contour, detected by my Edge Detection model, of the upper instance was not closed. To train the Edge Completion network edge images with not-closed contours are required, with a mask defining the region of missing edges. I generated the edge images with not-closed contours with an unpaired CycleGAN [110], which is able to learn features of the images. Typically this network is used for learning RGB features. My idea was that features of edge images of a given type of object could also be learned by this architecture. I used the aGT edge images and the predictions of the Edge Detection model on real frames containing occluding instances. Thus, the generated not-closed contours are similar to the output of the edge detection, and the corresponding (augmented) ground-truth edge images are provided by construction. The mask of the edge region is based on the foreground mask of the instances and the generated edge image with not-closed contours.

As opposed to the inpainting technique in the original application of the [95] network, I apply no masks on the RGB input. During training the missing contour is available based on the corresponding aGT image, therefore I modified the loss function of the discriminator network. The loss of the foreground and the loss of the missing contour contribute to the overall loss equally. The Edge Completion model, from training the Edge Completion Network as described above, further reduces the frames where the upper instance has non-closed contour.

DOI:10.15774/PPKE.ITK.2023.008

THESIS II.3. I created a segmentation algorithm that combines detected edges and detected body part regions from an existing unsupervised segmentation model to provide instance segmentation.

I used the Edge Detection and Completion models as follows. I estimated the foreground mask and generated edge images. If it was a single connected region then the algorithm of Edge Completion was invoked.

In the case of a single connected foreground region only the *body parts* method [56] was used. If the foreground mask had more than one region then we combined it with the results of the *body parts* method. An initial labeling is created for the regions created by the detected edges, after noise reduction. Edge-defined regions and regions from body part detection are assigned to each other based on the overlaps. This labeling provides the anchor regions of the watershed algorithm [115], applied within the foreground region. The outcome is the *per-frame segmentation*. The overview of the main steps is shown in Figure 4.3.

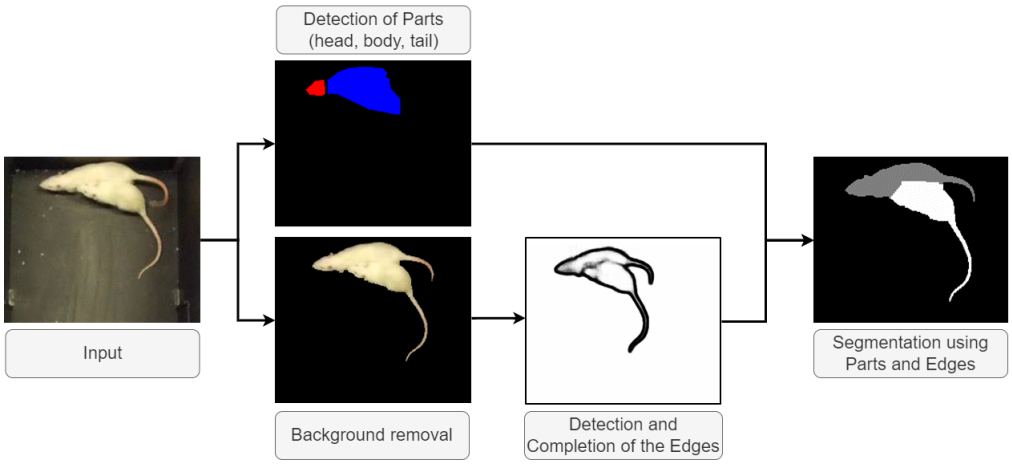

Figure 4.3: Sketch of the test pipeline for a single frame. Our aim is to maximize the segmentation precision within the frame to provide a strong basis for tracking. Body parts and edges are predicted and edge completion methods are invoked. A pre-processing provides the foreground masks and removes the background from the frame. A post-processing module combines the information and predicts the segmentation of the instances separately to enable identity tracking. Note the error in the last subfigure: the head of one of the animals is mislabeled. However, tracking remains error-free.

DOI:10.15774/PPKE.ITK.2023.008

Per-frame segmentations were connected by a propagation algorithm [57] to measure the reliability of tracking based on the segmentation. I compared my results to three state-of-the-art identity tracking methods of similar kinds. No trajectory switches occurred for my method, whereas the competitive methods made several mistakes, and are outperformed in identity tracking and instance segmentation of unmarked rats in real-world laboratory video recordings. Results are presented in Table 4.2.

Related publication of the Author: [A2]

The codes related to the theses are available at `https://github.com/g-h-anna/phd-diss-code`.

DOI:10.15774/PPKE.ITK.2023.008

Table 4.2: Comparison of segmentation-based trajectory tracking of methods which do not require prior data annotation, on our test data of 3 600 frames. TS denotes the track switches metric from [57], which measures the number of ID switches during tracking. Lost ID: there are less than two different foreground labels in the frame. For ToxTrac and idtracker, per frame segmentation masks are not available. Superscript 1 marks that the result is computed by means of the GUIs of the cited method, the best we could do for comparisons. BIPED-TL is the model trained with the augmented dataset with transfer learning on the original BIPED model [93]. Gray highlight: Mean of the Intersection over Union (IoU) values for all frames. For benchmark measures, we used the same ones as in [57].

ID Tracking Results							
Approach	**Num. of TS.**	**Num. of Frames with Lost IDs**	**IoU Mean**	**IoU & F Mean**	**IoU Recall**	**F Mean**	**F Recall**
ToxTrac [78]	9	267[1]	N/A	N/A	N/A	N/A	N/A
idtracker [76]	8	1055[1]	N/A	N/A	N/A	N/A	N/A
idtracker.ai [77]	10	1485	0.5556	0.604	0.59	0.652	0.746
BIPED & Parts	4	2	0.833	0.871	0.978	0.908	0.985
SEPARATS & Parts	0	0	0.846	0.883	0.994	0.921	1.000
BIPED-TL & Parts	0	0	0.845	0.883	0.994	0.921	0.999

DOI:10.15774/PPKE.ITK.2023.008

4.3 Application of the Results

The proposed deep pipeline of highly similar instance segmentation during occlusions outperformed similar approaches in segmentation and tracking quality on the annotated 3 600 images. After extending training data with a brightness-related color augmentation, it also tracked the instances without any track switches on a 224 577-frame dataset of four videos without annotated segmentations. Although this corresponds to a generalized edge detection model that required still no human annotation to deliver reliable tracking for a different setup (using another type of background or even instances that look differently), there are scenarios when a different approach can lead to a better solution. Deep models provide the highest prediction quality for data with the most similar distribution to the training data [124].Therefore, for a new data set of frames from a video clip containing consecutive scenes, which can be considered identically distributed data, we can expect the best prediction quality if the training images are most similar to the frames and, as such, have a similar distribution. We can make use of the great advantage of the pipeline that it generates training data fully automatically if provided with a set of frames of occluding and a set of frames with non-occluding instances, thus, requiring minimal human time and attention to obtain a training set adapted to the current input. With training data obtained through synthetic data generation and automatic labeling, we can train the edge detection model on highly similar images as those in the input video to get high-quality predicted segmentation. Therefore the proposed pipeline is expected to be well applicable for tasks to provide instance segmentation for id-tracking multiple, unmarked, sometimes occluding, highly similar instances with constant background and fixed distance from a fixed upper camera, if light conditions allow for the observation of visual properties (i.e. shadows) between the animals during occlusions, with a low amount of human attention and effort required.

Particle filtering is applied in multiple fields where the Kalman filter is suboptimal for state estimation due to the nonlinearity of the state dynamics and non-Gaussian

DOI:10.15774/PPKE.ITK.2023.008

noise. The proposed GPU Cellular Particle Filter retains the advantage of the original particle filtering being not limited to a specific domain. Be it image processing, autonomous driving, robotics, or any field where the particle filtering approach would lead to a solution, the proposed method can be exploited to achieve the same prediction quality with a more efficient, parallel adaption. The proposed adapted filter can be easily applied for one and two-dimensional inputs (as shown in the evaluation section for the two benchmark models) and can be modified to deal with higher dimensions. In our proposed method, the information sharing ratio is tunable and may be modulated adaptively. Therefore, it broadens the range of options, than using a predefined information share value to find the optimal share ratio range among particles to achieve the lowest error at the highest speed. For tasks that use detections generated on the GPU device by a deep model and particle filtering is required for tracking due to the noise or other characteristics, the GPU-adapted Cellular Particle Filter can be applied with a comparable estimation quality of the CPU single-threaded PF, but spare the time-consuming data transfer between GPU and CPU. Compared to the Fermi architecture with a compute capability of 2.x available by the time of the research, the Ampere GPU architecture introduced in 2020 with a compute capability of 8.x, offers significantly higher efficiency and speed. While retaining the main concept of the algorithm, the performance of Cellular Particle Filter on GPU would be increased due to several reasons, such as the changes regarding the shared memory. The faster memory access decreases the overhead of data transfers and computations within the shared memory, and shared memory size is three times larger on an Ampere device [125] than on a Fermi [126] thus, can reduce the number of global memory accesses for synchronization of the particles. Moreover, the memory bandwidth between memory types is also significantly increased in Ampere architecture compared to Fermi architecture, not only on-device but also the PCIe-v4 buses double the speed between CPU and GPU, memory bandwidth is 1.6TB/s (for A100 40GB)–3TB/s (for H100) compared to 192GB/s of Fermi. The proposed method enables parallel execution on modern GPUs, just as it did on architectures available 10 years ago. In addition to the

DOI:10.15774/PPKE.ITK.2023.008

numerical acceleration resulting from architectural advancements the implementation of the GPU CPF algorithm on current hardware is expected to maintain a similar magnitude difference compared to the implementation of the sequential CPU PF algorithm.

In my current and future research, I aim to reduce human time in multi-animal annotation and tracking. In my approach, deep models are trained or fine-tuned on synthetically generated training samples and, based on the deep models and traditional computer vision algorithms, I propose id-tracking for multiple ruff instances in side-view recordings with several occluding positions in collaboration with the Max Planck Institute for Biological Intelligence.

DOI:10.15774/PPKE.ITK.2023.008

Acknowledgements

First of all I would like to express my gratitude to my supervisors. I would like to thank György Cserey that he supported me to take the first steps towards research, and I could always rely on his positive and supportive personality, which helped me face and overcome the challenges that arose. His unwavering encouragement and belief in me gave me the confidence to persevere.

I am very grateful to Kristóf Karacs for his constant guidance and support. His insights, encouragement, and our weekly meetings and consultations have been invaluable to me. He has helped me as a mentor to find my way, feel capable, and grow over the years regarding both research and education. Throughout my work, his approachable and collegial attitude has been a constant source of motivation, providing me with a sense of support that I deeply appreciate. I look forward to continuing to learn from and work with him in the future.

I would like to express my gratitude to Professor András Lőrincz, who gave me the opportunity to join the highly inspiring *Neural Information Processing Group* (NIPG) group. Without his time, advice, and professional guidance the results of Thesis group II. would not have been accomplished. He gave me considerable freedom in my research direction, while still always demanded- and guided me toward high standards. I am grateful for his thoughtful support in not only guiding me through research questions but also considering my long-term goals, and I look forward for continuing our collaboration.

I would like to thank Judit Nyékyné Gaizler, Péter Szolgay, and Kristóf Iván deans for all their support, and the opportunity to carry out my research at the University. I am very grateful to Gábor Tornai, for all professional advice during my CPF research, and to András Horváth for his help in discussing the particle filter algorithms. It has been a fantastic opportunity to work together with the members of the NIPG group, especially

DOI:10.15774/PPKE.ITK.2023.008

with László Kopácsi, Áron Fóthi, Zsombor Fülöp, Viktor Varga, Kinga Faragó, Gergő Ungvári, Ervin Téglás, Kevin Hartyáni, Milán Szász, Bálint Kovács. The last two years, with all our joint projects and discussions, were very valuable for me.

I would like to thank the Department of Artificial Intelligence of ELTE Faculty of Informatics for all support and that I could develop my research using the NIPG servers. The support of ELTE Faculty of Science is also kindly acknowledged. I am grateful to the Max Planck Institute for Biological Intelligence for giving me the opportunity to extend and continue my research in our collaboration.

Special thanks go to the administrative members of the Faculty of Information Technology and Bionics, for all technical support and help in administrative tasks, especially to Dr. Tivadarné Vida and Mária Babiczné Rácz.

Last and most importantly, I can not express my level of gratitude to my family. The support of my husband Andris, and the joy of my children, Julcsi and Marci, was indispensable to me, as was their perseverance and patience. I would like to thank my mother and father for all the inspiration, motivation, and faith in me. The countless hours they spent looking after my children were essential for me to work on my research.

DOI:10.15774/PPKE.ITK.2023.008

Appendix A

List of independent citations of article [A1] of the author*

1. Cole, C. B., Machine Learning Methods for next Generation Sequencing Data: Applications to MLL-AF4 Leukemia and Demographic Inference, *University of Oxford, PhD thesis*, 2021.

2. Spyridon Patmanidis, Alexandros Charalampidis, George P. Papavassilopoulos, Tumor Growth Modeling: State Estimation with Maximum Likelihood and Particle Filtering, *28th Mediterranean Conference on Control and Automation (MED), IEEE*, 2020, 144-149. 10.1109/MED48518.2020.9183193.

3. Havard Heitlo Holm, Martin Sætra, Peter Jan Van Leeuwen Massively parallel implicit equal-weights particle filter for ocean drift trajectory forecasting,*Journal of Computational Physics: X. 6. 100053.*, 2020, 10.1016/j.jcpx.2020.100053.

4. Dan Crisan, Joaquin Miguez, Gonzalo Ricardo Ríos Muñoz, On the performance of parallelisation schemes for particle filtering, *EURASIP Journal on Advances in Signal Processing*, 2018.05, doi:10.1186/s13634-018-0552-x

5. David Jáuregui, Patrick Horain, Real-time 3D motion capture by monocular vision and virtual rendering., *Machine Vision and Applications, 28.*, 2017., 10.1007/s00138-017-0861-3.

6. Sile Hu, Qiaosheng Zhang, Jing Wang, Zhe Chen, Real-time particle filtering and

*Date of list: 2023. April 3.

DOI:10.15774/PPKE.ITK.2023.008

smoothing algorithms for detecting abrupt changes in neural ensemble spike activity, *Journal of Neurophysiology, 119(4)*, 2017, 10.1152/jn.00684.2017.

7. Grigorios Mingas, Algorithms and architectures for MCMC acceleration in FPGAs, *Electrical and Electronic Engineering PhD theses, Imperial College London*, 2015, DOI:10.25560/31572

8. Achutegui, Katrin et al., *A simple scheme for the parallelization of particle filters and its application to the tracking of complex stochastic systems*, 2014, DOI:10.48550/arXiv.1407.8071

DOI:10.15774/PPKE.ITK.2023.008

Appendix B

Appendix for CPF GPU

B.1 Modification of NVIDIA SDK Mersenne Twister

NVIDIA SDK provides an implementation of Mersenne Twister (MT) [127, 128], which apparently exposes an attainable solution. Still, we observed that the generated distribution is inappropriate for a small set (hundreds or thousands) of numbers, and is primarily admissible for around 2 million numbers and above. As we mentioned earlier in relation to GPU memory management, the main operations are performed in shared memory, thus, random number generator is required to comply with shared memory array size. Consequently, the original NVIDA SDK MT is not feasible for our implementation. Figure B.1 shows the difference of the histogram of the MT generated numbers compared to the histogram of a same number of random values from MATLAB.

To achieve an adequate distribution for the resampling, we made the modifications as follows based on [127] and our empirical experiences. The degree of recursion was changed from 19 to 397, the middle term from 9 to 624, the shift value u from 12 to 11 based on the original values from [127]. Besides, in the tempering transformation we used the originally defined masks instead of the loaded ones from a predefined file, with hexadecimal values in the bitwise operations. For each thread the first element of the state array is calculated with a thread and current time based seed value based on the thread ID and the current system time. The final value is calculated with initializing on the first element of the bit vector for each thread.

With the above modifications, we achieved an acceptable uniform distribution from

DOI:10.15774/PPKE.ITK.2023.008

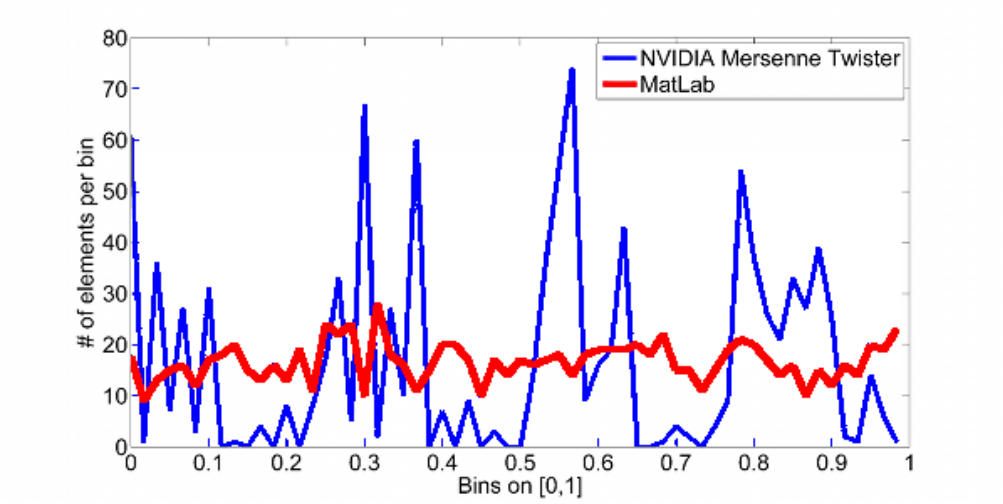

Figure B.1: Mersenne Twister uniform distribution (gray) histogram compared to MATLAB uniform distribution (black) with 60 histogram bins on 1,000 random numbers. We can see significant spikes in the histogram of NVIDIA Mersenne Twister which would introduce bias to the resampling. Therefore, it is not suitable for our purpose.

the Mersenne Twister (MT), which is illustrated on Figure B.2 compared to the former MT and the MATLAB distributions.

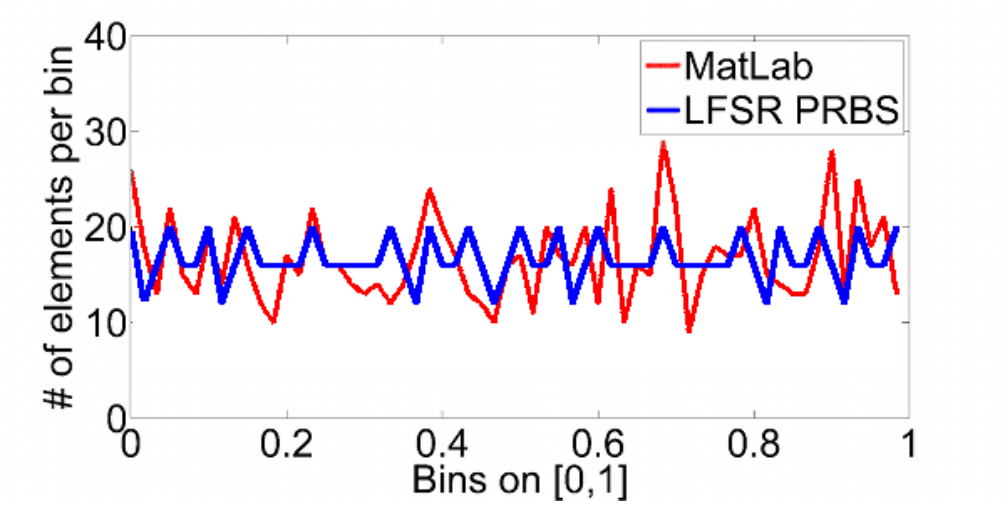

Figure B.2: A total of 1,000 random number were generated with the modified implementation of Mersenne Twister compared to MATLAB and original Mersenne Twister distributions on 60 bins. Due to new characteristics, now we have a suitable random number generator.

DOI:10.15774/PPKE.ITK.2023.008

List of author's publications

List of journal publications

[A1] **Anna Gelencsér-Horváth** et al. "Fast, parallel implementation of particle filtering on the GPU architecture". In: *EURASIP J. Adv. Signal Process.* 2013 (2013), p. 148. DOI: 10.1186/1687-6180-2013-148. URL: https://doi.org/10.1186/1687-6180-2013-148.

[A2] **Anna Gelencsér-Horváth** et al. "Tracking Highly Similar Rat Instances under Heavy Occlusions: An Unsupervised Deep Generative Pipeline". In: *Journal of Imaging* 8.4 (2022). ISSN: 2313-433X. DOI: 10.3390/jimaging8040109. URL: https://www.mdpi.com/2313-433X/8/4/109.

List of proceedings publications

[A3] **Anna Horváth**. "Cellular Particle Filter on GPU". In: vol. 2011-2012 Academic year. 2012, pp. 145–148. URL: https://itk.ppke.hu/uploads/articles/159/file/phd_proceedings_2012.pdf.

[A4] **Anna Horváth**. "Region-merging based on contour-structure of clusters in oversegmented image". In: vol. 2012-2013 Academic year. 2013, pp. 79–81. URL: https://itk.ppke.hu/uploads/articles/159/file/Mini_simposium_2013.pdf.

[A5] **Anna Gelencsér-Horváth**. "Using contour geometry as a merging cue in oversegmented images". In: 2014, pp. 91–94. URL: https://itk.ppke.hu/uploads/articles/159/file/Mini_simposium_2014.pdf.

DOI:10.15774/PPKE.ITK.2023.008

References

[1] Kyungnam Kim and Larry Davis. "Multi-camera Tracking and Segmentation of Occluded People on Ground Plane Using Search-Guided Particle Filtering". In: May 2006, pp. 98–109. ISBN: 978-3-540-33836-9. DOI: 10.1007/11744078_8.

[2] Antonio Brunetti et al. "Computer vision and deep learning techniques for pedestrian detection and tracking: A survey". In: *Neurocomputing* Vol.300 (2018), pp. 17–33. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.01.092. URL: https://www.sciencedirect.com/science/article/pii/S092523121830290X.

[3] Qiwu Luo et al. "Automated Visual Defect Classification for Flat Steel Surface: A Survey". In: *IEEE Transactions on Instrumentation and Measurement* 69 (Dec. 2020), pp. 9329–9349. DOI: 10.1109/TIM.2020.3030167.

[4] Cheng-Ju Kuo et al. "Improving Defect Inspection Quality of Deep-Learning Network in Dense Beans by Using Hough Circle Transform for Coffee Industry". In: Oct. 2019, pp. 798–805. DOI: 10.1109/SMC.2019.8914175.

[5] D.J. Withey and Z.J. Koles. "Medical Image Segmentation: Methods and Software". In: *2007 Joint Meeting of the 6th International Symposium on Noninvasive Functional Source Imaging of the Brain and Heart and the International Conference on Functional Biomedical Imaging*. 2007, pp. 140–143. DOI: 10.1109/NFSI-ICFBI.2007.4387709.

[6] Geert Litjens et al. "A survey on deep learning in medical image analysis". In: *Medical Image Analysis* 42 (2017), pp. 60–88. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2017.07.005. URL: https://www.sciencedirect.com/science/article/pii/S1361841517301135.

[7] Donka Angelova and Lyudmila Mihaylova. "Contour segmentation in 2D ultrasound medical images with particle filtering". In: *Mach. Vis. Appl.* 22 (May 2011), pp. 551–561. DOI: 10.1007/s00138-010-0261-4.

[8] Priyanka Malhotra et al. "Deep Neural Networks for Medical Image Segmentation". In: *Journal of Healthcare Engineering* 2022 (Mar. 2022), pp. 1–15. DOI: 10.1155/2022/9580991.

[9] Adrian Dalca. "Segmentation of Nerve Bundles and Ganglia in Spine MRI using Particle Filters". PhD thesis. Mar. 2012.

[10] Sandor Szabo and Marta Alexy. "Practical Aspects of Weight Measurement Using Image Processing Methods in Waterfowl Production". In: *Agriculture* 12.11 (2022). ISSN: 2077-0472. URL: https://www.mdpi.com/2077-0472/12/11/1869.

[11] Suresh Neethirajan. "ChickTrack – A quantitative tracking tool for measuring chicken activity". In: *Measurement* 191 (2022), p. 110819. ISSN: 0263-2241. DOI: https://doi.org/10.1016/j.measurement.2022.110819. URL: https://www.sciencedirect.com/science/article/pii/S0263224122001154.

DOI:10.15774/PPKE.ITK.2023.008

[12] André Ferreira et al. “Deep learning-based methods for individual recognition in small birds”. In: *Methods in Ecology and Evolution* 11 (July 2020). DOI: 10.1111/2041-210x.13436.

[13] Abhishek Gupta et al. “Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues”. In: *Array* 10 (2021), p. 100057. ISSN: 2590-0056. DOI: https://doi.org/10.1016/j.array.2021.100057. URL: https://www.sciencedirect.com/science/article/pii/S2590005621000059.

[14] Nitin Kanagaraj et al. “Deep learning using computer vision in self driving cars for lane and traffic sign detection”. In: *International Journal of System Assurance Engineering and Management* 12.6 (Dec. 2021), pp. 1011–1025. DOI: 10.1007/s13198-021-01127. URL: https://ideas.repec.org/a/spr/ijsaem/v12y2021i6d10.1007_s13198-021-01127-6.html.

[15] Bhaskar Barua et al. “A Self-Driving Car Implementation using Computer Vision for Detection and Navigation”. In: *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*. 2019, pp. 271–274. DOI: 10.1109/ICCS45141.2019.9065627.

[16] P. del Moral. “Nonlinear Filtering Using Random Particles”. In: *Theory of Probability & Its Applications* 40.4 (1996), pp. 690–701. DOI: 10.1137/1140078. eprint: https://doi.org/10.1137/1140078. URL: https://doi.org/10.1137/1140078.

[17] M.S. Arulampalam et al. “A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking”. In: *Signal Processing, IEEE Transactions on* 50.2 (2002), pp. 174–188. ISSN: 1053-587X.

[18] Liangjie Jia et al. “Low-SNR Infrared Point Target Detection and Tracking via Saliency-Guided Double-Stage Particle Filter”. In: *Sensors* 22.7 (2022). ISSN: 1424-8220. DOI: 10.3390/s22072791. URL: https://www.mdpi.com/1424-8220/22/7/2791.

[19] Y. Ephraim and N. Merhav. “Hidden markov processes”. In: *Information Theory, IEEE Transactions on* 48.6 (2002), pp. 1518–1569. ISSN: 0018-9448.

[20] Rudolph Emil Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[21] Dan Simon. “Kalman Filtering with State Constraints: A Survey of Linear and Nonlinear Algorithms”. In: *Control Theory & Applications, IET* 4 (Sept. 2010), pp. 1303–1318. DOI: 10.1049/iet-cta.2009.0032.

[22] Nuramin Fitri Aminuddin et al. “Hungarian-Particle Filtering Based Segmentation for On-Road Visual Vehicle Detection and Tracking”. In: *2022 IEEE 4th Global Conference on Life Sciences and Technologies (LifeTech)*. 2022, pp. 600–603. DOI: 10.1109/LifeTech53646.2022.9754928.

[23] H.F. Lopes and R.S. Tsay. “Particle filters and Bayesian inference in financial econometrics”. In: *Journal of Forecasting* 30.1 (2011), pp. 168–209. ISSN: 1099-131X.

DOI:10.15774/PPKE.ITK.2023.008

[24] S. Chib, F. Nardari, and N. Shephard. “Markov chain Monte Carlo methods for stochastic volatility models”. In: *Journal of Econometrics* 108.2 (2002), pp. 281–316. ISSN: 0304-4076.

[25] Petar Djuric and Mónica Bugallo. “Particle filtering for high-dimensional systems”. In: Dec. 2013, pp. 352–355. ISBN: 978-1-4673-3144-9. DOI: 10.1109/CAMSAP.2013.6714080.

[26] N. Azzabou, N. Paragios, and F. Guichard. “Image reconstruction using particle filters and multiple hypotheses testing”. In: *Image Processing, IEEE Transactions on* 19.5 (2010), pp. 1181–1190. ISSN: 1057-7149.

[27] J. Czyz, B. Ristic, and B. Macq. “A particle filter for joint detection and tracking of color objects”. In: *Image and Vision Computing* 25.8 (2007), pp. 1271–1281. ISSN: 0262-8856.

[28] F. Gustafsson et al. “Particle filters for positioning, navigation, and tracking”. In: *Signal Processing, IEEE Transactions on* 50.2 (2002), pp. 425–437. ISSN: 1053-587X.

[29] M. de Bruijne and M. Nielsen. “Image segmentation by shape particle filtering”. In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Vol. 3. 2004, 722–725 Vol.3. DOI: 10.1109/ICPR.2004.1334630.

[30] Kang-Hyun Jo. “A Novel Particle Filter Implementation for a Multiple-Vehicle Detection and Tracking System Using Tail Light Segmentation”. In: *International Journal of Control Automation and Systems* 11 (June 2013), pp. 577–585. DOI: 10.1007/s12555-012-0353-1.

[31] Hamd Abdelali et al. “Visual Vehicle Tracking via Deep Learning and Particle Filter”. In: Oct. 2020, pp. 517–526. ISBN: 978-981-15-6047-7. DOI: 10.1007/978-981-15-6048-4_45.

[32] A. Doucet and A. M. Johansen. “A tutorial on particle filtering and smoothing: fifteen years later”. In: *Oxford Handbook of Nonlinear Filtering* (2008), pp. 4–6.

[33] Gustaf Hendeby, Rickard Karlsson, and Fredrik Gustafsson. “Particle Filtering: The Need for Speed”. In: *EURASIP J. Adv. Signal Process* 2010 (Feb. 2010). ISSN: 1110-8657. DOI: 10.1155/2010/181403. URL: https://doi.org/10.1155/2010/181403.

[34] M. Bolic, P.M. Djuric, and Sangjin Hong. “Resampling algorithms and architectures for distributed particle filters”. In: *Signal Processing, IEEE Transactions on* 53.7 (2005), pp. 2442–2450. ISSN: 1053-587X. DOI: 10.1109/TSP.2005.849185.

[35] C.Y. Chu et al. “Multi-prediction particle filter for efficient parallelized implementation”. In: *EURASIP Journal on Advances in Signal Processing* 2011.1 (2011), pp. 1–13.

[36] Anthony Lee et al. “On the Utility of Graphics Cards to Perform Massively Parallel Simulation of Advanced Monte Carlo Methods”. In: *Journal of Computational and Graphical Statistics* 19.4 (2010). PMID: 22003276, pp. 769–789. DOI: 10.1198/jcgs.2010.10039. eprint: https://doi.org/10.1198/jcgs.2010.10039. URL: https://doi.org/10.1198/jcgs.2010.10039.

DOI:10.15774/PPKE.ITK.2023.008

[37] Raúl Cabido et al. “Multiscale and local search methods for real time region tracking with particle filters: local search driven by adaptive scale estimation on GPUs”. In: *Machine Vision and Applications* 21 (2008), pp. 43–58.

[38] Kazuhiro Otsuka and Junji Yamato. “Fast and Robust Face Tracking for Analyzing Multiparty Face-to-Face Meetings”. In: *Machine Learning for Multimodal Interaction*. Ed. by Andrei Popescu-Belis and Rainer Stiefelhagen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 14–25. ISBN: 978-3-540-85853-9.

[39] Min-An Chao et al. “Efficient parallelized particle filter design on CUDA”. In: *2010 IEEE Workshop On Signal Processing Systems* (2010), pp. 299–304.

[40] Mehdi Chitchian et al. “Distributed Computation Particle Filters on GPU Architectures for Real-Time Control Applications”. In: *IEEE Transactions on Control Systems Technology* 21.6 (2013), pp. 2224–2238. DOI: 10.1109/TCST.2012.2234749.

[41] Olivier Brun, Vincent Teulière, and Jean-Marie Garcia. “Parallel Particle Filtering”. In: *J. Parallel Distributed Comput.* 62 (2002), pp. 1186–1202.

[42] A.S. Bashi et al. “Distributed implementations of particle filters”. In: *Sixth International Conference of Information Fusion, 2003. Proceedings of the*. Vol. 2. 2003, pp. 1164–1171. DOI: 10.1109/ICIF.2003.177369.

[43] Vasileios Belagiannis et al. “Segmentation Based Particle Filtering for Real-Time 2D Object Tracking”. In: vol. 7575. Oct. 2012, pp. 842–855. ISBN: 978-3-642-33764-2. DOI: 10.1007/978-3-642-33765-9_60.

[44] J.V. Candy. “Bootstrap Particle Filtering”. In: *Signal Processing Magazine, IEEE* 24.4 (July 2007), pp. 73–85. ISSN: 1053-5888. DOI: 10.1109/MSP.2007.4286566.

[45] A. Horvath and M. Rasonyi. “Topographic implementation of particle filters on cellular processor arrays”. In: *Elsevier Signal Processing* (2013). DOI: 10.1016/j.sigpro.2012.11.025.

[46] L. Chua and L. Yang. “Cellular Neural Networks: Theory”. In: *IEEE Trans. on Circuits and Systems* 35(10) (1988), pp. 1257–1272.

[47] Dorin Comaniciu and Peter Meer. “Meer, P.: Mean shift: A Robust Approach Toward Feature Space Analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence 24(5), 603-619”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24 (June 2002), pp. 603–619. DOI: 10.1109/34.1000236.

[48] Li Liu et al. “Deep Learning for Generic Object Detection: A Survey”. In: *International Journal of Computer Vision* 128.2 (Feb. 2020), pp. 261–318. ISSN: 1573-1405. DOI: 10.1007/s11263-019-01247-4. URL: https://doi.org/10.1007/s11263-019-01247-4.

[49] Tobias Glasmachers. “Limits of End-to-End Learning”. In: *CoRR* abs/1704.08305 (2017). arXiv: 1704.08305. URL: http://arxiv.org/abs/1704.08305.

[50] Aston Zhang et al. “Dive into deep learning”. In: *arXiv preprint arXiv:2106.11342* (2021).

DOI:10.15774/PPKE.ITK.2023.008

[51] Gartner. *The 4 Trends That Prevail on the Gartner Hype Cycle for AI*. https://www.gartner.com/en/articles/the-4-trends-that-prevail-on-the-gartner-hype-cycle-for-ai-2021. Online; accessed 7-Dec-2021. 2021.

[52] Debmalya Biswas. *Compositional AI: Fusion of AI/ML Services*. Mar. 2021. URL: https://www.researchgate.net/publication/351037326_Compositional_AI_Fusion_of_AIML_Services.

[53] Mary Johnson. "Laboratory Mice and Rats". In: *Materials and Methods* 2 (Oct. 2012). DOI: 10.13070/mm.en.2.113.

[54] Elizabeth Bryda. "The Mighty Mouse: The Impact of Rodents on Advances in Biomedical Research". In: *Missouri medicine* 110 (July 2013), pp. 207–11.

[55] Michael Festing, Paraskevi Diamanti, and J.A. Turton. "Strain differences in haematological response to chloroamphenicol succinate in mice: Implications for toxicological research". In: *Food and chemical toxicology : an international journal published for the British Industrial Biological Research Association* 39 (May 2001), pp. 375–83. DOI: 10.1016/S0278-6915(00)00149-6.

[56] László Kopácsi, Áron Fóthi, and András Lőrincz. "A Self-Supervised Method for Body Part Segmentation and Keypoint Detection of Rat Images". In: *Annales Univ. Sci. Budapest., Sect. Comp*. Vol. 53. 2021.

[57] László Kopácsi et al. "RATS: Robust Automated Tracking and Segmentation of Similar Instances". In: *Artificial Neural Networks and Machine Learning – ICANN 2021*. Springer, 2021, pp. 507–518. ISBN: 978-3-030-86365-4. DOI: 10.1007/978-3-030-86365-4_41.

[58] Rickard Karlsson Gustaf Hendeby and Frederick Gustafsson. "Particle Filtering: The Need for Speed". In: *Advances in Signal Processing, EURASIP Journal on* June.Article ID 181403 (2010). DOI: 10.1155/2010/181403.

[59] L. Murray. "GPU acceleration of the particle filter: The Metropolis resampler". In: *Distributed machine learning and sparse representation with massive data-sets*. Jan. 2011.

[60] D. Salmond Gordon N. and A.F.M. Smith. "Novel approach to nonlinear/nonGaussian Bayesian state estimation". In: *IEE Proceedings* F-140 (1993), pp. 107–113.

[61] J.H. Halton. "Sequential Monte Carlo". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 58 (1962), pp. 57–78.

[62] X.-L. Hu, T. B. Schön, and L. Ljung. "A General Convergence Result for Particle Filtering". In: *IEEE Transactions on Signal Processing* 59.7 (July 2011), pp. 3424–3429.

[63] *NVIDIA CUDA Programming Guide*. [Accessed 27-Feb-2013]. URL: http://developer.nvidia.com/object/cuda.html.

[64] G.J. Tornai, G. Cserey, and I. Pappas. "Fast DRR generation for 2D to 3D registration on GPUs." In: *Medical physics* 39.8 (2012), p. 4795.

DOI:10.15774/PPKE.ITK.2023.008

[65] Vadim Demchik. "Pseudo-random number generators for Monte Carlo simulations on ATI Graphics Processing Units". In: *Computer Physics Communications* 182.3 (2011), pp. 692–705. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2010.12.008. URL: https://www.sciencedirect.com/science/article/pii/S0010465510004868.

[66] W. B. Langdon. "A Fast High Quality Pseudo Random Number Generator for NVidia CUDA". In: GECCO '09. Montreal, Québec, Canada: Association for Computing Machinery, 2009, pp. 2511–2514. ISBN: 9781605585055. DOI: 10.1145/1570256.1570353. URL: https://doi.org/10.1145/1570256.1570353.

[67] Markus Manssen, Martin Weigel, and Alexander Hartmann. "Random number generators for massively parallel simulations on GPU". In: *The European Physical Journal Special Topics* 210 (Apr. 2012). DOI: 10.1140/epjst/e2012-01637-8.

[68] Bradley P. Carlin, Nicholas G. Polson, and David S. Stoffer. "A Monte Carlo Approach to Nonnormal and Nonlinear State-Space Modeling". In: *Journal of the American Statistical Association* 87.418 (1992), pp. 493–500.

[69] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". In: *IEEE Proceedings F, Radar and Signal Processing* 140.2 (1993), pp. 107–113. DOI: 10.1049/ip-f-2.1993.0015.

[70] G. Kitagawa. "Monte Carlo filter and smoother for non-Gaussian nonlinear state space models". In: *Journal of computational and graphical statistics* 5.1 (1996), pp. 1–25.

[71] James P. Bohnslav et al. "DeepEthogram, a machine learning pipeline for supervised behavior classification from raw pixels". In: *eLife* 10 (Sept. 2021), e63377. ISSN: 2050-084X. DOI: 10.7554/eLife.63377.

[72] Jessy Lauer et al. "Multi-animal pose estimation and tracking with DeepLabCut". In: *bioRxiv 2021.04.30.442096* (2021). DOI: 10.1101/2021.04.30.442096. eprint: https://www.biorxiv.org/content/early/2021/04/30/2021.04.30.442096.full.pdf.

[73] Talmo Pereira et al. "SLEAP: Multi-animal pose tracking". en. In: *bioRxiv* (Sept. 2020), p. 2020.08.31.276246.

[74] Simon RO Nilsson et al. "Simple Behavioral Analysis (SimBA)–an open source toolkit for computer classification of complex social behaviors in experimental animals". In: *BioRxiv* (2020).

[75] Oliver Sturman et al. "Deep learning-based behavioral analysis reaches human accuracy and is capable of outperforming commercial solutions". In: *Neuropsychopharmacology* 45.11 (2020), pp. 1942–1952.

[76] Alfonso Pérez-Escudero et al. "IdTracker: Tracking individuals in a group by automatic identification of unmarked animals". In: *Nature methods* 11 (June 2014). DOI: 10.1038/nmeth.2994.

[77] Francisco Romero-Ferrero et al. "Idtracker. ai: tracking all individuals in small or large collectives of unmarked animals". In: *Nature methods* 16.2 (2019), pp. 179–182.

DOI:10.15774/PPKE.ITK.2023.008

[78] Alvaro Rodriguez et al. “ToxId: an efficient algorithm to solve occlusions when tracking multiple animals”. In: *Scientific reports* 7.1 (2017), pp. 1–8.

[79] David Eilam and Ilan Golani. “Home base behavior of rats (Rattus norvegicus) exploring a novel environment.” In: *Behavioural brain research 34.3, Elsevier* (2016), pp. 199–211.

[80] Veronica Panadeiro et al. “A review of 28 free animal-tracking software applications: Current features and limitations”. In: *Lab animal* (2021), pp. 1–9.

[81] Alvaro Rodriguez et al. “ToxTrac: a fast and robust software for tracking organisms”. In: *Methods in Ecology and Evolution* 9.3 (2018), pp. 460–464.

[82] Xiaodong Lv et al. “A Robust Real-Time Detecting and Tracking Framework for Multiple Kinds of Unmarked Object”. In: *Sensors* 20.1 (2020). ISSN: 1424-8220. DOI: 10.3390/s20010002. URL: https://www.mdpi.com/1424-8220/20/1/2.

[83] Kuo-Kun Tseng et al. “A fast instance segmentation with one-stage multi-task deep neural network for autonomous driving”. In: *Computers & Electrical Engineering* 93 (2021), p. 107194. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2021.107194. URL: https://www.sciencedirect.com/science/article/pii/S0045790621001920.

[84] Kaiming He et al. “Mask R-CNN”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.

[85] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3431–3440. DOI: 10.1109/CVPR.2015.7298965.

[86] Liang-Chieh Chen et al. “DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs”. In: *arXiv preprint arXiv:1606.00915* (2016).

[87] Roland S. Zimmermann and Julien N. Siems. “Faster training of Mask R-CNN by focusing on instance boundaries”. In: *Computer Vision and Image Understanding* 188 (2019), p. 102795. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2019.102795. URL: https://www.sciencedirect.com/science/article/pii/S1077314218303059.

[88] Hao Chen et al. “DCAN: Deep contour-aware networks for object instance segmentation from histology images”. In: *Medical Image Analysis* 36 (2017), pp. 135–146. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2016.11.004. URL: https://www.sciencedirect.com/science/article/pii/S1361841516302043.

[89] Yunong Tian et al. “Instance segmentation of apple flowers using the improved mask R–CNN model”. In: *Biosystems Engineering* 193 (2020), pp. 264–278. ISSN: 1537-5110. DOI: https://doi.org/10.1016/j.biosystemseng.2020.03.008.

[90] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker. “Design of an image edge detection filter using the Sobel operator”. In: *IEEE Journal of solid-state circuits* 23.2 (1988), pp. 358–367.

DOI:10.15774/PPKE.ITK.2023.008

[91] J. Canny. “A Computational Approach to Edge Detection”. In: *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms*. Ed. by M. A. Fischler and O. Firschein. Los Altos, CA.: Kaufmann, 1987, pp. 184–203.

[92] Justin Lazarow et al. “Learning Instance Occlusion for Panoptic Segmentation”. In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[93] Xavier Soria Poma, Edgar Riba, and Angel Sappa. “Dense Extreme Inception Network: Towards a Robust CNN Model for Edge Detection”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Mar. 2020.

[94] Saining Xie and Zhuowen Tu. “Holistically-Nested Edge Detection”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1395–1403. DOI: 10.1109/ICCV.2015.164.

[95] Kamyar Nazeri et al. “EdgeConnect: Structure Guided Image Inpainting using Edge Prediction”. In: *The IEEE International Conference on Computer Vision (ICCV) Workshops*. Oct. 2019.

[96] Yuxin Wu et al. *Detectron2, 2019.*, https://github.com/facebookresearch/detectron2.

[97] Yuxin Ma et al. “Self-supervised vessel segmentation via adversarial learning”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 7536–7545.

[98] Dahye Kim and Byung-Woo Hong. “Unsupervised segmentation incorporating shape prior via generative adversarial networks”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 7324–7334.

[99] Antoine Saporta et al. “Multi-target adversarial frameworks for domain adaptation in semantic segmentation”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 9072–9081.

[100] Rameen Abdal et al. “Labels4free: Unsupervised segmentation using stylegan”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 13970–13979.

[101] Tom Monnier et al. “Unsupervised layered image decomposition into object prototypes”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 8640–8650.

[102] Mengde Xu et al. “End-to-end semi-supervised object detection with soft teacher”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 3060–3069.

[103] Kaihua Zhang et al. “Deep transport network for unsupervised video object segmentation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 8781–8790.

[104] Honglin Chen et al. “Unsupervised segmentation in real-world images via spelke object inference”. In: *European Conference on Computer Vision*. Springer. 2022, pp. 719–735.

DOI:10.15774/PPKE.ITK.2023.008

[105] Mathilde Caron et al. “Emerging properties in self-supervised vision transformers”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 9650–9660.

[106] Pengwan Yang et al. “Less than Few: Self-Shot Video Instance Segmentation”. In: *European Conference on Computer Vision*. Springer. 2022, pp. 449–466.

[107] C. J. Van Rijsbergen. *Information Retrieval*. 2nd. Butterworth-Heinemann, 1979.

[108] Áron Fóthi et al. “Multi Object Tracking for Similar Instances: A Hybrid Architecture”. In: *International Conference on Neural Information Processing*. Springer, Cham, Nov. 2020, pp. 436–447. ISBN: 978-3-030-63829-0. DOI: 10.1007/978-3-030-63830-6_37.

[109] Alexandru Telea. “An Image Inpainting Technique Based on the Fast Marching Method”. In: *Journal of Graphics Tools* 9 (Jan. 2004). DOI: 10.1080/10867651.2004.10487596.

[110] Jun-Yan Zhu et al. “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networkss”. In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. 2017.

[111] Ian Goodfellow et al. “Generative Adversarial Networks”. In: *Advances in Neural Information Processing Systems* 3 (June 2014). DOI: 10.1145/3422622.

[112] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, 2013. URL: https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8%5C&qid=1336857747%5C&sr=8-2.

[113] Elizabeth Million. *The Hadamard Product*. 2007.

[114] Stéfan van der Walt et al. “scikit-image: image processing in Python”. In: *PeerJ* 2 (June 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: https://doi.org/10.7717/peerj.453.

[115] Anton S. Kornilov and Ilia V. Safonov. “An Overview of Watershed Algorithm Implementations in Open Source Libraries”. In: *Journal of Imaging* 4.10 (2018). ISSN: 2313-433X. DOI: 10.3390/jimaging4100123. URL: https://www.mdpi.com/2313-433X/4/10/123.

[116] Shihao Jiang et al. “Learning to Estimate Hidden Motions with Global Motion Aggregation”. In: *arXiv preprint arXiv:2104.02409* (2021).

[117] Alex Bewley et al. “Simple online and realtime tracking”. In: *2016 IEEE international conference on image processing (ICIP)*. IEEE. 2016, pp. 3464–3468.

[118] Mark Everingham et al. *The PASCAL Visual Object Classes (VOC) challenge*. 2010.

[119] Federico Perazzi et al. “A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. URL: https://github.com/fperazzi/davis.

[120] Xavier Soria Poma et al. “Dense Extreme Inception Network for Edge Detection”. In: *CoRR* abs/2112.02250 (2021). arXiv: 2112.02250. URL: https://arxiv.org/abs/2112.02250.

DOI:10.15774/PPKE.ITK.2023.008

[121] Mahmoud Afifi and Michael S. Brown. "What Else Can Fool Deep Learning? Addressing Color Constancy Errors on Deep Neural Network Performance". In: *The IEEE International Conference on Computer Vision (ICCV)*. Oct. 2019.

[122] O. Morris, M. Lee, and A. Constantinides. "A unified method for segmentation and edge detection using graph theory". In: *ICASSP '86. IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 11. 1986, pp. 2051–2054. DOI: 10.1109/ICASSP.1986.1168866.

[123] *Apptainer: Application containers*. [Last accessed 16-Jan-2023]. URL: https://apptainer.org/.

[124] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[125] NVIDIA. *Ampere architecture paper*. "Accessed 2023-02-08". URL: https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

[126] *NVIDIA Fermi architecture paper*. "Accessed 2023-02-08". URL: https://www.nvidia.com/content/PDF/fermi%5C_white%5C_papers/NVIDIA%5C_Fermi%5C_Compute%5C_Architecture%5C_Whitepaper.pdf.

[127] M. Matsumoto and T. Nishimura. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator". In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8.1 (1998), pp. 3–30. ISSN: 1049-3301.

[128] Victor Podlozhnyuk. *Parallel Mersenne Twister*. [Accessed 27-Feb-2013]. June 2007. URL: http://developer.download.nvidia.com/compute/cuda/2_2/sdk/website/projects/MersenneTwister/doc/MersenneTwister.pdf.