Universität Bremen

Master-Thesis

Random forests for tracking on ultrasonic images

An object tracking algorithm for medical ultrasonic images, based on a machine learning method

Supervisor

Prof. Dr. Gabriel Zachmann
Prof. Dr.-Ing. Udo Frese

Author

Frank Alexander Ihle
Matriklenummer: 3010158
Systems Engineering M. Sc.

Abstract

This thesis describes two possibilities for object tracking on medical ultrasonic image data, which was provided by the CLUST2015 challenge. The first method classifies with a Random Forest arbitrary pixel, whether they are part of the object or not. By calculating the center of the retrieved object pixel mass, the actual location is estimated. Therefore, the algorithm generates autonomous datasets and labels. The latter method's design consists of two main parts that together estimate coordinates of a selected tissue in each frame. The first is a *rough* tracker to approximate the vicinity of the object. The subsequent *fine* tracker improves this first guess, by searching for the exact location within this constricted area. The rough prediction is based on Random Forests that estimate the current sector of a learned trajectory, which is retrieved during a few breathing cycles from the start, without supervision. To capture variations in the object's movement, the algorithm adaptively learns new possible locations and shifts the learned positions whenever the trajectory drifts and also artificially interpolates non-ultrasonic data to increase algorithm's robustness. The thesis concludes with an evaluation and the results of the CLUST challenge.

Acknowledgement

All praise, honour and glory to my Lord Jesus Christ for the many blessings I have encountered, during the times at the university and especially in Bremen.

For all the support and patience I would like to express my deepest gratitude to my family and friends for supporting me, that were there from the beginning and the ones I have found along the way, while doing the unexpected.

My sincere thanks to my thesis supervisor Prof. Dr. Gabriel Zachmann of the CGVR at Universität Bremen. The door to Prof. Zachmann Office was always open, whenever I had a question about my research or writing. Through this opportunity I was able to gain a lot of knowledge in this exciting field of research and for that I am grateful.

I would also like to acknowledge Prof. Dr.-Ing. Udo Frese of the agebv at Universität Bremen as the second reader of this thesis, for the valuable comments on this thesis and the ideas I could gather during his lecture in Echtzeitbildverarbeitung.

Many special thanks goes to Christoph Schröder for the support during this period, especially for the invested time, the topics about machine learning and having an eye on the challenge paper.

Disclaimer

I hereby declare that this thesis is my own original work and has not been submitted before to any institution for assessment purposes. Further, I have acknowledged all sources used and have cited these in the reference section.

Frank Alexander Ihle

Date

Abbreviations

CPU Central processing unit

CNN Convolutional neural network

FP false positive

FN false negative

GPU Graphics processing unit

GT ground truth

ML machine learning

ORB Oriented FAST and Rotated BRIEF

RF Random Forest

ROI region of interest

ROC receiver operating characteristics

SVM support vector machine

SIFT Scale-invariant feature transform

TP true positive

TN true negative

Contents

List of Tables

List of Algorithms

1

Introduction

Medical engineering has reached a level of advance, that nowadays machines and robots have established themselves in this area as useful tools, besides the usual computers. For quite some time, many utilities are used for surveillance and diagnosis, starting from small devices like electrical pulse monitors to large apparatuses for the magnetic resonance tomography. In the meantime, there are already first machines to support surgical operations for very fine and fragile areas, that shall apply the incision on human organs instead of the doctor himself. They stand out, especially for smaller wounds and a quicker operation time compared to a human's performance. The application of such instruments is intended, because it is desired to increase the operational precision and quality, as well as to filter out disturbances, which are caused for instance by a trembling hand from the surgeon [1][2]. Therefore, it is appropriate to transfer the incision's execution to a machine. Such a machine is required to withstand any disturbances in order to set the surgical cut at the correct position. This is the reason why it needs to recognize and adapt itself to the patient's body movements. Prominent examples of such robot-based medical systems are:

Cyberknife	- a robot with six degrees of freedom for local precision radiation [3].
Robodoc	- for applications in the area of hip replacement surgery [4].
da Vinci	- its usage primarily is dedicated to the gynecologic, as well as the urologic area [5].

Admittedly, both the doctor and the machine underly external influences. Nevertheless at the current state of the art, a machine is able to respond faster to any of such occurrences. With the ability to control incoming disturbances, the impact is reduced and thus the operational quality improved. For this reason, the device depends especially on up-to-date information about the patient's status. This includes in particular, the current movement of affected inner body parts and the point of the breathing cycle currently reached by the patient (this helps to prepare for a forthcoming change of direction of organ's movements).

Ultrasonic scanners in the medical field are commonly utilized for diagnosis. Due to their nature of operating without surgical cuts or other invasive actions, portability and their comparably low costs, made the ultrasonic imaging attractive for tissue motion analysis or tracking. The difficulty lies in the image processing, where a general vague, noisy and sometimes smooth appearance of the field of view impedes algorithms to automatically follow a region of interest (ROI).

Since it is applied especially in the medical field, algorithm failures are to avoid in any case. Therefore, not only a precise localization of the object is important, but also the machine security is required as one of the main aspects, which makes working with such tools feasible in first place. Because otherwise, an unsteady signal could cause the robot to malfunction.

1.1 Motivation

In order to perform reliably, it is necessary for these medical support robots, to receive a robust signal, that makes it possible for them to adjust to the current circumstances. An ideal signal is defined by a high resolution and also an uninterrupted, steady data output. The higher the precision is demanded for an operation, the higher the related resolution needs to be, since it is the basis for further processing. Because the robot depends on this kind of information in the end, an interruption would cause the machine to work *blindly*. This is why the need to focus on providing a robust signal is crucial.

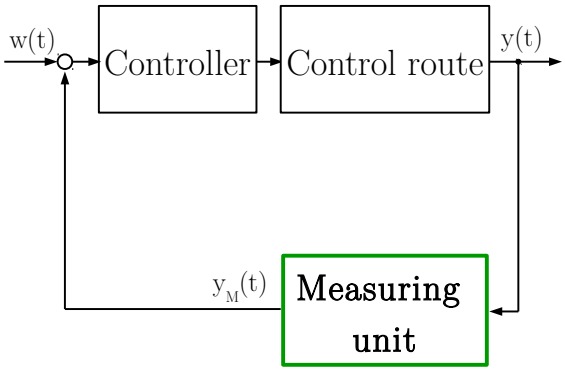


Figure 1: Schematic construction of the control loop, as part of a medical support robot. The application of the presented algorithm shall be used as measuring unit.

To give a better impression, this sensor element could represent a measuring unit in the control loop of the medical robot. Schematically this is shown in figure 1.

The aim of this thesis is therefore developing a tissue-tracking algorithm on image recordings of an ultrasound scanner in general. In particular this shall be achieved with the help of a machine learning method called *Random Forest*, which is based on *Decision Trees*.

For the usage of this algorithm beyond laboratory conditions, it is needed to optimize the processing speed, because a high update rate promotes the precision maximization of the controller.

Many approaches have been published in [6] describing already precise methods, but their robustness usually relies on the consecutiveness of frames. If the algorithm depends for instance on the vicinity of the last ROI estimate and a bigger jump in position occurred (e.g. by a system freeze), then this search area is left out automatically and it might be possible to lose the object, which aborts the tracking in general.

Smaller regions of interest (ROI) with the size of a fraction from a whole image could serve as a tracking object. These have the advantage that one is able to obtain a more rapid processing time, because of their minimal size and the possibility that comes with it, to leave out the other parts of the image. Figure 2 illustrates such an image record with a region that (or a similar region) could be processed and tracked.

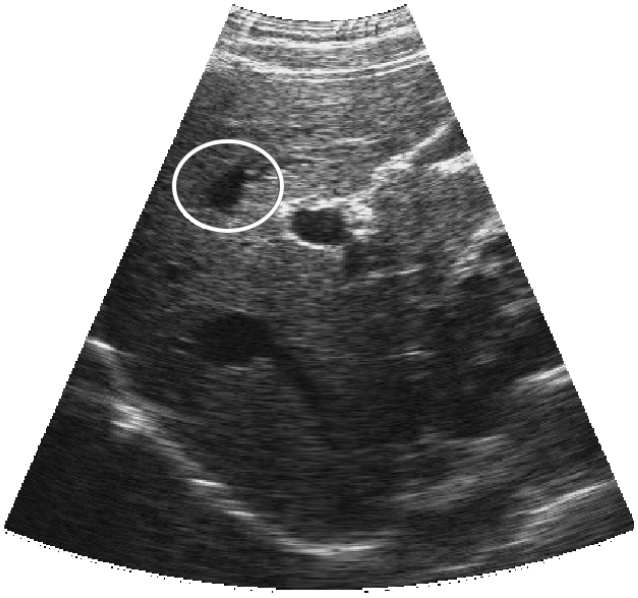


Figure 2: Example of an ultrasound image, which is used in this thesis. Encircled is a possible region, traceable in a consecutive image sequence.

1.2 State of the art

For this special kind of image analysis a research competition has been established known by the name MICCAI CLUST2015 [6]. In this challenge multiple approaches that solve the issue differently, are compared to each other. Below a few with excellent results are briefly summarized.

Optical Flow is applied on two consecutive ultrasonic images, which have been binarized beforehand and processed with a median-filter. Then key points are generated in both of the pictures and the distance is calculated, how far two corresponding points are drifted apart. On the one hand, this is achieved with a comparison of the previous frame, and on the other hand this matching was also done with a reference frame, to increase robustness. Mean tracking error $\approx$ 1.05 mm, takes around 20-70 ms per image record [7].

With the help of a kernelized correlation filter with multiple small image cutouts along the object's trajectory. The cutouts are created during an initial run. For distinguishing fore- and background a discrete Fourier transform is applied. Mean tracking error $\approx$ 1.09 mm, takes $\approx$ 223-155 ms per image record [8].

A convolutional neural network is utilized to determine the motion vector and to recognize the current position in multiple smaller image cutouts, along the objects trajectory. The position statement is selected between these two points whose distance of the recently calculated annotation and the ones, who have been captured initially are the shortest. Mean tracking error $\approx$ 2.83 mm, takes $\approx$ 200 ms per image record [9].

Another approach is based on a common tracking method, that iteratively estimates the next position in three steps. At first, a smaller image section around the ROI is generated, subsequently this section should be rediscovered with SIFT, where its new center is then defined as the current one. For verification purposes, this center is compared with the initial position from step one. Mean tracking error $\approx$ 0.92 mm, takes $\approx$ 0.2 s per image record [10].

Besides this challenge, there are alternatives even to optical methods. One approach addresses an electromagnetic tracking, where a coil is generating a magnetic field. On the receiver sided antenna varies depended from its position the frequency, phase and amplitude of the current [11].

1.3 Goals of this work

Concluding the following points can be determined as key points of this thesis:

- Object tracking with tissues on ultrasonic images.
- Evaluation of the algorithm's parameters.
- Minimal processing time per image, to ensure a possibility for a real-time scenario.
- Minimal user effort (i.e.: as algorithm input only a few sample ROI).
- Output: current position specification for a given frame.

2

General fundamentals

This chapter will outline the theoretical foundations of the thesis, such as the data that is to be processed, as well as the machine learning (ML) basics and concludes with a discussion about the chosen method.

2.1 Ultrasonic image records

As well known, ultrasonic signals are used by animals for instance bats, it is used for underwater range finding and needless to say for the well known foetal imaging during pregnancies. Furthermore, a bit less known examples can be found in engineering (e.g.: welding of plastic and metallic materials), biology (rupturing cell walls to release contents), dentistry (cleaning and drilling) or geology (localization of oil and mineral deposits) [12, p.1,5].

Medical sonography applications, as it is used in this thesis, is based on high-frequency acoustic pulses of a short duration. Theses signals are emitted from a transceiver, situated on the patients outer skin. This device is sender and receiver of the signal at the same time. When ultrasound is propagated through the body, it encounters different physical boundaries, from where the pulse is then reflected (or *back-scattered*) at various body elements in different directions. This yields in a series of echoes, which in turn are detected again by the transceiver. The time difference between emitting and receiving holds the information about how deep the signal could enter the body. Specific features of the reflecting structures (e.g.: composition, compressibility and density) have influence on the signals strength or amplitude. Basically ultrasound images represent a record from an echo of the emitted signal, assembled line by line, where each line stands for an individual emitted pulse. The signals amplitude is converted into a gray scale intensity response that together build up the desired image. Where a very high intensity (displayed as bright) corresponds to a strong reflection, whereas a low intensity comes from an absence of the signal [12, p.10]. A higher frequency results in a better image but goes by the flaw of a belittled depth of penetration [13].

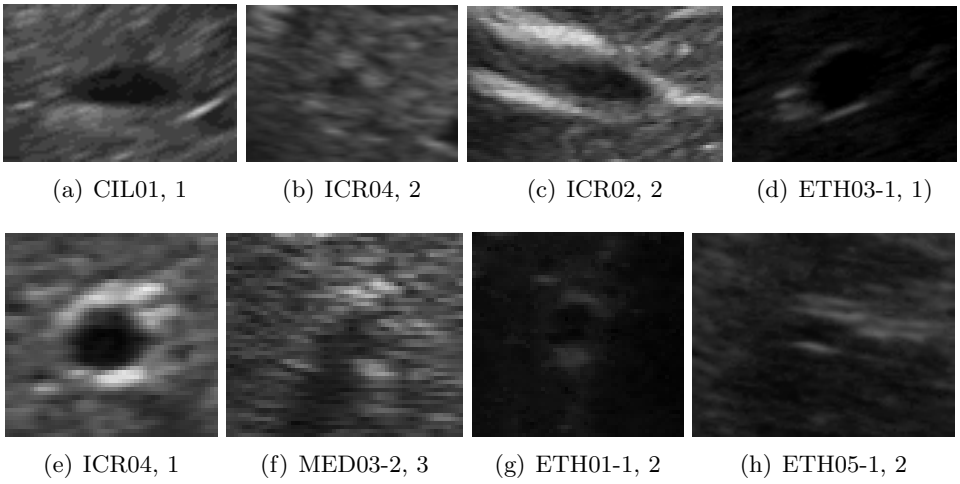
2.1.1 Traceable regions and difficulties

The algorithm is developed on basis of the training section of the CLUST2015 dataset. To give an insight a summary of traceable ROIs are shown in figure 3 with representative examples.

Because of their different appearances, shapes and orientation it does not come in easy to define a general behavior to distinguish between *object* and its *environment*. Moreover there is a variety of difficulties that could or could not occur along the tracking process. A good algorithm not only is able to minimize the distance of the estimated and actual position, it also comes along with an user-friendly interface. Ideally, the human interaction is reduced to start the program with the localization of the desired object, which is used to autonomously process the ultrasonic image data (i.e. the user is prevented from custom settings). For developing of such a generic but robust approach, an overview is

desired including all possible ROI, even if they were considered as rarity.

Hence, below in figure 3 a summary is given about possible appearances of the ROIs, which the algorithm needs to be capable to handle. In the figure, the captions contain the sequence name and their landmark ID, separated by a comma respectively.



(a) CIL01, 1 (b) ICR04, 2 (c) ICR02, 2 (d) ETH03-1, 1)

(e) ICR04, 1 (f) MED03-2, 3 (g) ETH01-1, 2 (h) ETH05-1, 2

Figure 3: Examples from the training dataset that was used to develop the algorithm.

Besides the impressions about the traceable objects from above, there are further attributes, specifically for the ROI itself:

- Variation in orientation.
- Variation in intensities.
- Variation in shape.
- Variation in size.
- Occlusion / ROI is barely recognizable.
- Fast and slow movement.
- Trajectory might drift, thus leaving its path.
- Periodic behavior cannot be guaranteed.

Other considerable issues for the tracking system:

- Speckles that blur the vision.
- Unusual jumps (or jerks) along the trajectory.
- Temporary change of full image brightness.
- Image disturbances generated by the system.
- Processing of non-ultrasonic data, when the ROI reached the image margin.
- Back- and foreground of the ROI could be more or less the same.

2.1.2 Image granulation

Capturing the ultrasound signal also underlies a result-distorting noise. Certainly there is the usual quantification noise, derived from converting analog to digital units. But this technology additionally has to deal with a special kind of noise that is known by the name *speckle*.

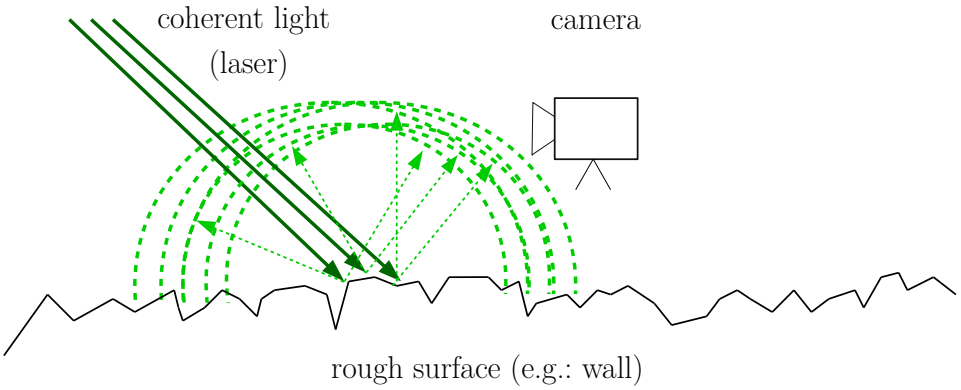


Figure 4: Speckle pattern generation with laser.

This specific effect is generated when ultrasound waves are reflected differently and several entities intersect at one point. In other words the amplitudes of the signals are added at a particular location, which yields into constructive or destructive interference and because of the amount of involved elements, a barely predictable pattern is generated [14]. This behavior is well known in other disciplines as well, for example in the laser technology. Coherent emitted light is reflected by a surface (e.g. a wall) whose different reflection interfere with each other [15, p. 131]. To give a vivid example, the speckle patterns are schematically represented for the laser granulation in figure 4. This can easily be observed when emitting a common laser pointer onto a wall, the granular spots depict the mentioned interference.

Depending on its application it is either a disturbance or can be used constructively as gauging tool. For instance in communication engineering, speckle patterns are considered as complication for the transmission as it causes fluctuations in a received signal. Also laser speckle working with is a tool can retrieve the information about materials unevenness [16, p.39 f.] and is used for shape measurements [17, p.18 ff.]. One constructive application of this phenomenon on ultrasonic image data is tissue characterization [18].

It appears as a granular pattern on ultrasonic image data, corresponding to homogeneous tissues. Fully developed speckle patterns arise when a critical scatter concentration exceeds [19]. Scattering can be subdivided into three groups: specular, diffusive and diffractive. Specular scattering is generated when the object is big compared to the wavelength, whereas diffusive scattering is the opposite, when the object compared to the wavelength is small. Diffractive scattering occurs when the size of the object is in between the two types described above. Usually a tissue is aggregated of small sub wavelength point scatters, a sum of many small reflections with uniformly distributed phases. It is important not to confuse speckle with random noise, it is rather a reproducible pattern, if it is captured under similar conditions [20, p. 6].

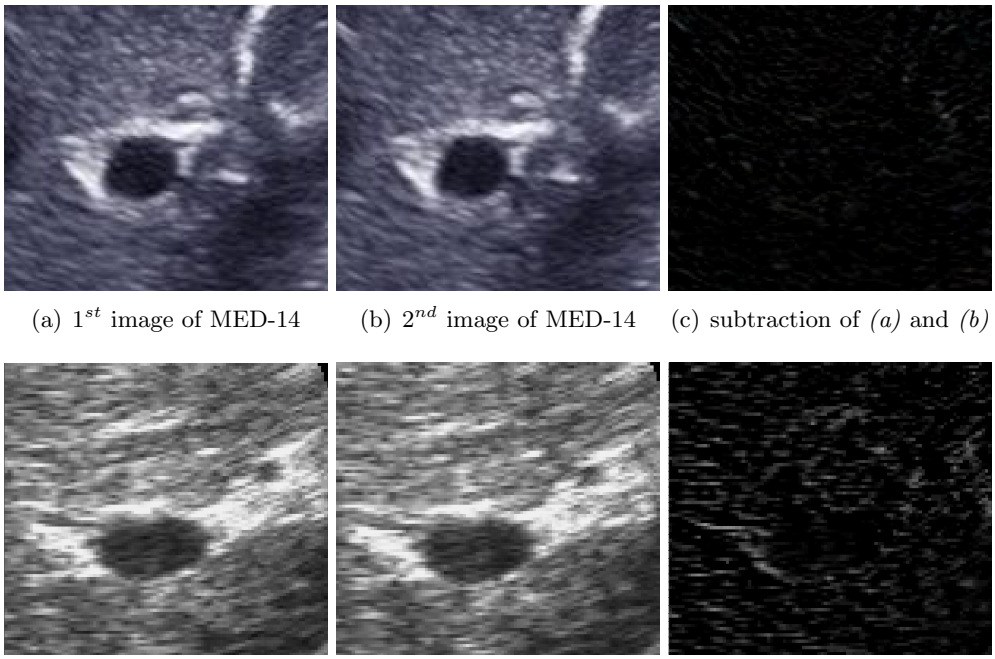
However it takes influence on the image precision, since its appearance blurs the image data. For a better imagination *speckle* on ultrasonic images resemble the *salt and pepper* noise. In the end this disturbance generates a yielding border between object and background, that makes it harder to define the target as a whole.

Additionally because the speckle pattern vary even on bare motions, thus a image subtraction of two consecutive records is highly impeded, which is a known technique for detecting moving parts in the area of image processing. If y and x stand for the coordinates, P_1 and P_2 of two consecutive images, then a resulting image Q can be calculated by

$$\sum_{y=0}^{heigth} \sum_{x=0}^{width} Q(y,x) = P_1(y,x) - P_2(y,x) \tag{1}$$

What happens in equation 1 is very simple. If a pixel at one position has the same intensity level in P_1 and P_2 the value will be subtracted to zero. However if an investigated part of the image has moved in one of the pictures the intensity level usually changes as well, this yields a result other than zero. So every pixel in the subtracted image has a different value than zero, it could be considered as a moving part between the two frames. This technique is useful in robot vision, where the machine has to detect moving parts in a static area.

However, applied on ultrasonic image data the results are not promising, an example is shown in the figures 5. One problem unfortunately is, the movement of the background is detected too, which makes it hard to determine the object's movement by machine, since the back- and foreground can have a different velocity. Another issue is the different look and feel of the resulting pattern. Where in (f), for a human hardly the margin of a ROI could be perceived, (c) in turn is showing a totally different behavior. So in conclusion, simple approaches for this tracking purpose are likely to fail.



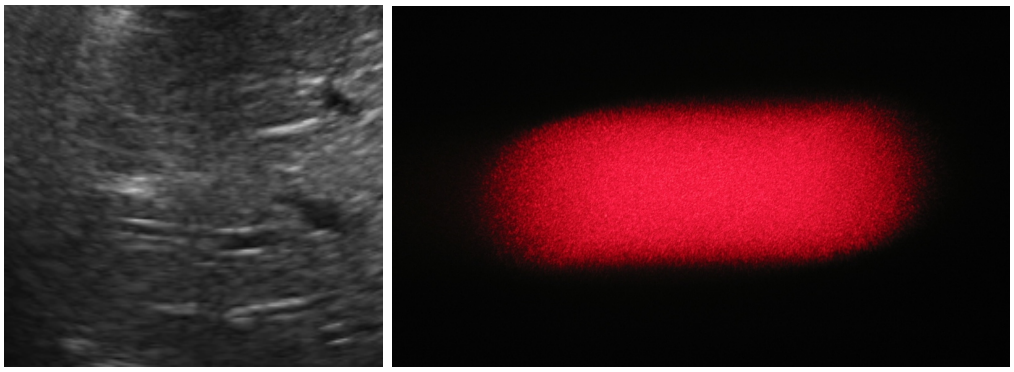
(a) 1^{st} image of MED-14 (b) 2^{nd} image of MED-14 (c) subtraction of *(a)* and *(b)*

(d) 1^{st} image of MED-02 (e) 2^{nd} image of MED-02 (f) subtraction of *(c)* and *(d)*

Figure 5: The result of a image subtraction of consecutive image data.

Fig. 6 shows an example of an ultra sonic image with speckles, with the *look-and-feel* of salt and pepper noise to the left, whereas on the right speckles of a coherent light is

displayed: a laser granulation.



(a) Speckles on a ultra sonic image. (Excerpt from sequence *ICR-01*)

(b) Speckle spots of a laser on white paper [21].

Figure 6: Different appearances of speckle pattern.

2.2 Machine learning

The following pages contain a basic overview about machine learning. Besides the naming and common methods, also a few utilities will be handled, that will be used later on.

2.2.1 Terminology

First of all a basic terminology is introduced, which is common in the machine learning community.

Classification and Regression

In classification, a sample is analyzed and distinguished what fits the most, of a given amount of classes (e.g.: whether a client is credit worthy or not). Whereas regression retrieves numerical values of a wide range (e.g.: for calculating a price of a car, when input parameters are brand, construction year, power, mileage etc.) [22, p. 4f, 9]. In case of tissue tracking a classifier could identify pixels, if they belong either to the *object* or not, while in regression a trajectory could be learned and their x- and y coordinates with two regressors predicted respectively, independent from each other.

Features

An issue that shall be analyzed can be described with different properties, in this field of technology these are known by the name features. They appear in different types of values, either deterministic (for instance using *color* with n-possible states, as there are n colors one likes to take into account), or as continuous number where there is theoretically an infinite amount of states (e.g.: population or temperature) [23, p. 2]. For instance

when analyzing the meteorology, practical features are humidity, temperature and air pressure, whereas shadiness, soil type and frequency of irrigation constitute indicators for agriculture.

Feature-vector and dataset

As mentioned above features are used to describe an issue one likes to inspect. The combination of multiple features for a particular situation is called a feature-vector. This combination solely describes just one case of the whole problem, linking many vectors of the same structure but with individual states (i.e.: the same kind of features but with different values) lead to a dataset. A dataset may or may not contain a label per feature-vector, depending on the desired algorithm (e.g.: labels are mandatory for classification, whereas they are not in clustering). The dataset is then used to fit the chosen algorithm to the present case. It doesn't necessarily need to be big, it crucially depends on the variety of samples it has to deal with in future. However a sparse dataset, with only a part of the possible occurring cases will probably retrieve poor outcomes (where of course some methods in the same situation return better and others worse results). Good feature-vectors help to separate the classes, but the question remains about how to deal with vectors, that are identical except of the attached label. This may be handled individually when for instance all concerning vectors get discarded or an additional class *unknown* is introduced that prevents making false predictions.

Generally speaking a dataset could be composed for instance of audio signals, analog or digital sensor values, weather states, image data, stock prices, etc. Basically any kind of data can be processed, as long it is representable in a computer (i.e. reassembled in numbers). For instance, this thesis is working with gray scale ultrasonic images. Such a picture could be described as an array of intensity levels, where each cell stands for one pixel. These intensity levels then again are saved in a value of a specific data type. When this data type reserves e.g. 8 bits per pixel (implies that there are $2^8 = 256$ possible states), each image point could be set in a range of 254 different shades of gray, as one full black and one full white tone.

Training- and testing phase

An algorithm is required to obtain sufficient information before the testing phase, which is gathered in the dataset. During the training phase the algorithm is adjusting itself to a given situation. The actual application of a method takes place in the testing phase. A well prepared training phase is crucial for the success of a desired accurate prediction. However there are cases where the training is continued when testing has already begun, thus continuously optimizing the algorithm.

Over- and underfitting

One of the main ideas of machine learning is to inductively adjust the computer to predict unseen or untrained data correctly. But there is the *overfitting* that lowers the prediction quality in the testing phase. If this occurs, then the chosen method will learn

noise, extracted from the provided dataset, instead of the actual significant parts of the issue. On the contrary when a dataset consists too less samples, at least for one of the issue's particular behaviors, then it might learn with insufficient information and a different situation will be trained. This is known by the name *underfitting* resulting likewise in a lower prediction quality. *Generalization* is the term when it comes to describe how good a method handles the unseen data.

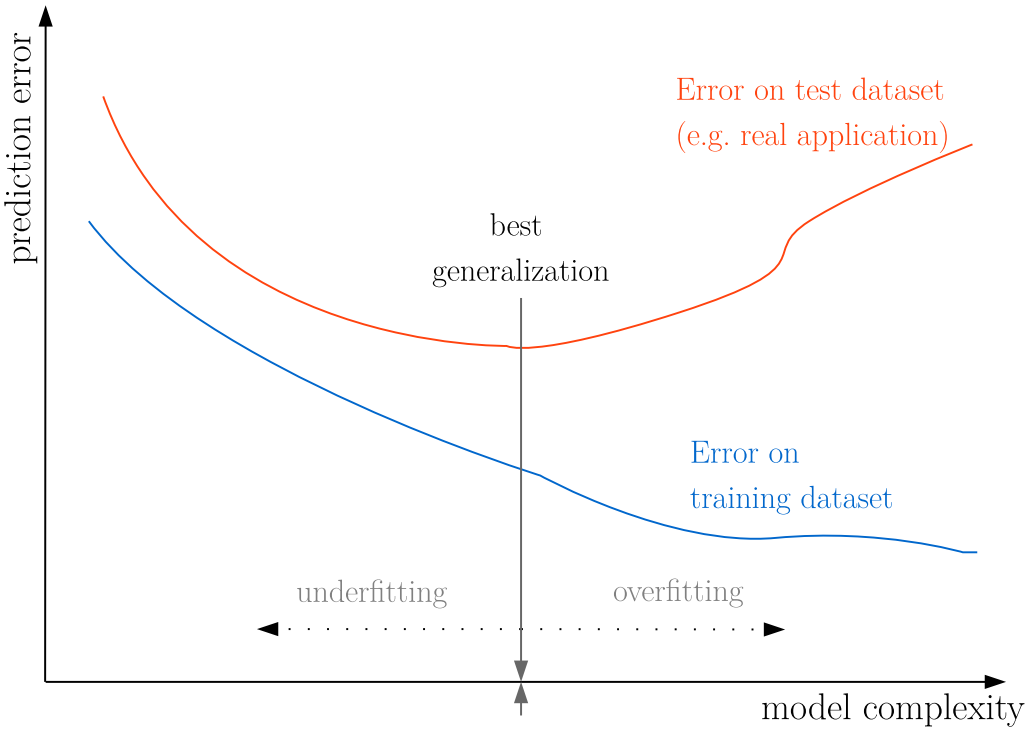


Figure 7: Visualization of general over- and underfitting in machine learning.

Figure 7 (cv. [24]) gives a graphical representation of the problem. During training (blue line) the classifier gets adjusted step by step, as its goal is to lower the error predictions. The difficulty is, the training- and testdataset are different from each other, and if the method's settings were adjusted on another set, the amount of false predictions may increase (as indicated by the red line). Therefore, it is desired to find the best position between *the algorithm has learned too less examples* and *the algorithm is training noise.*

2.2.2 Methodes

The foundation of the pursued *Random Forest (RF)* in this thesis is based on the *Decision Tree* technique. Both count to machine learning algorithms. In most of the times,

these methods consist of a training and a testing phase, in other words a particular problem first needs to be learned, before it can be analyzed to estimate arbitrary samples in th end. ML methods are often used, if a simple approach is failing, for instance because of ambiguous signal meaning in specific situations. As a solution a more intelligent concept is desired in order to remove the ambiguity .

Various approaches have been established in the area of ML. Choosing an appropriate method for a given problem, is often a compromise between prediction accuracy, processing time and hardware requirements. For example a lightweight Bayesian classifier's advantages are amongst others the simplicity, the learning and classification speed [25]. Where neural networks are more complex to develop and also the training time is comparably higher, but with a better accuracy on non-trivial problems in most of the cases.

Prominent machine learning examples are:

- SUPPORT VECTOR MACHINE (SVM) - try to define a function, that ignores cases if there are error estimates below a certain tolerance boundary but doesn't accept samples if this limit was exceeded, thus adjust its function appropriately [26].
- NEURAL NETWORKS - the idea is to reproduce a human or animal brain on a computer, to utilize its decision ability. It is basically trained with punishment and rewards [27, p. 483, 487].
- REINFORCEMENT LEARNING - similar to neural networks, it is based on punishment and rewards but performs without any examples. Instead, based on the current situation, different possible solutions are calculated and e.g. the one that promises the highest reward is chosen [28, p. 7].
- DECISION TREES / RANDOM FOREST - as explained in the chapters 2.3 and 2.4.

All these examples have in common that they depend on a dataset, where the meaning of each excerpt (*sample*) carries a description (usually designated as *label*). Such a dataset sometimes needs to be setup and arranged by a human, which may become a very time consuming procedure. This could be one reason, why not for every situation a labeled dataset can be provided. For this case, or if such a set simply isn't available when there is no history for a given problem, there is a subsection of ML, that performs with raw, unlabeled data. It is called *clustering* and well-known examples are:

- K-NEAREST NEIGHBORS - k-vectors have a certain localization, then iteratively the training samples are added to whose distance to these containers is the shortest. Repeatedly the samples swap from one position to another, as their vector localizations vary according to their contained data [29, p. 554 f.].
- HIERARCHICAL CLUSTERING - create groups where its instances are more similar to each other than in other groups, this is repeated with assembling similar groups layer by layer until a desired amount of groups is reached [22, p. 153 f.].

- DBSCAN - is a density based method to detect clusters of arbitrary shape and the noise [30].

Due to the chosen method is RF, a labeled dataset needs to be provided as input condition.

2.2.3 Utilities

For developing a robust machine learning algorithm it is important to have an insight on the impact if a certain parameter, the training data, or anything else is adjusted. There are a lot of different possibilities to measure whether the system has changed to a better level or not, tools especially for classification, regression and others. But not only when it comes to fine tuning the parameter settings, but to also extract a rough impression if a certain approach comes in handy or not. To give an overview, a few are be explained below, since they are being utilized in the algorithm's development.

2.2.3.1 Error measurement for deterministic cases

Many measuring tools for the classification are derived from a *class-confusion matrix*. Its structure is displayed in table 1 with TP = true positive, TN = true negative, FP = false positive, FN = false negative.

Table 1: Structure of the confusion matrix.

		Classified as	
		positive	negative
Labeled as	positive	TP	FN
	negative	FP	TN

Once a classifier (e.g.: a decision tree) is trained, it is verified with samples where the respective label is retained. For each sample, four cases could occur, separated in: right prediction (TP, TN) and false prediction (FP, FN). The following equations represent a selection of derived measurement units for an arbitrary classifier.

$$accuracy(\mathrm{TP}, \mathrm{TN}, \mathrm{FP}, \mathrm{FN}) = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{2a}$$

$$specificity(\mathrm{TN}, \mathrm{FP}) = \frac{\mathrm{TN}}{\mathrm{FP} + \mathrm{TN}} \tag{2b}$$

$$true\ positive\ rate\ /sensitivity(\mathrm{TP}, \mathrm{FN}) = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2c}$$

$$false\ positive\ rate(\mathrm{TN}, \mathrm{FP}) = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{2d}$$

A more vivid measurement is given with the receiver operating characteristics (ROC). Where a graph is visualized with the true-positive and the false-positive rate (as shown in equation 2c and 2d respectively). This shall help to design the classifier and find a good balance between two cases. Either a classifier is designed, where one can assume that each prediction is always correct but with the cost to retrieve rather less results. Or to receive as many results where a class could be predicted, with the drawback that not each result will be correct [31]. As an example, an artificial ROC is shown in fig 8, where each orange cross is the result of the same algorithm with different parameter settings, in this case 17 settings are evaluated. A cross in the very top-left corner means, the classifier was always correct, down-left: classifier predicts always true, up-right: classifier predicts always false.

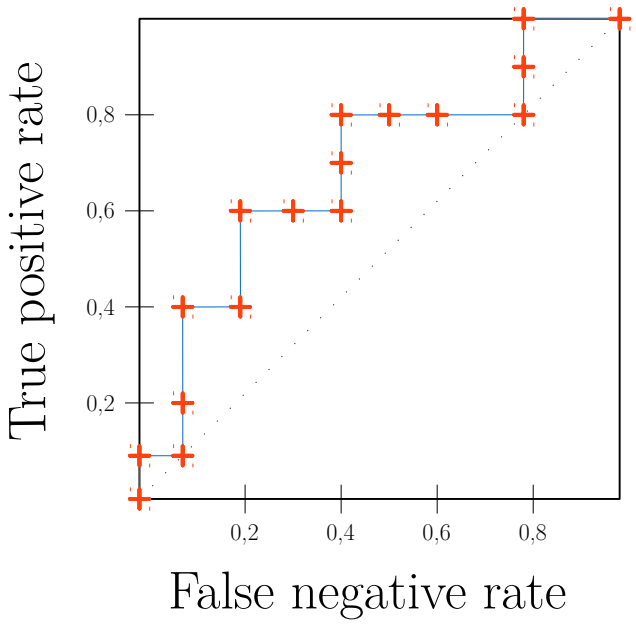


Figure 8: A sample ROC curve to adjust the parameter settings of a classifier.

2.2.3.2 Error measurements for continuous numeric cases

Scoring the precision of a continuous, numerical value needs special tools, compared to classification. It can be applied for regression tasks or for instance in coordinates estimation. Quite often only the distance between the predicted and actual result is compared. This yields two possible adjustments for this rating, independent from each other: *metric* and *distance*.

On a more distant view, in *metrics* all intermediate results are utilized for rating a given dataset (e.g. the position estimates from a sequence of ultrasonic images). It might be appropriate to normalize such results before checking between datasets that deal with another range of values, in order to create a validity for inter-comparison. For instance an (for error measurement) ideal, small ROI, with a size of 1x1 pixel has a very clear and unambiguous defined center for calculating a certain distance. Though, when its dimensions increase and the shape varies, it is not that easy to determine the exact coordinate anymore (e.g. it could be the pixel's centroid or just their averaged position). Subsequently three methods are briefly explained, which will be used in chapter 4 for evaluation, with n as the amount of comparable spots, $d(a, b)$ as the actual *distance* (positively a similarity or negatively the error) of two information containers a and b for a certain instance (e.g. the position of on a frame).

$$mean\ error : \bar{d}(a,b) = \frac{1}{n}\sum_{i=1}^{n} d_i(a,b) \tag{3a}$$

$$standard\ deviation : s = \sqrt{\frac{1}{n-1}\ \sum_{i=1}^{n}(d_i(a,b) - \bar{d}(a,b))^2} \tag{3b}$$

Equation 3a is focused on displaying the center of the error distribution, which is often not enough to have a profound insight of the current measurement. Therefore additional information can be extracted with 3b, since it provides the information how heavily the information was distributed around its center (3a). In other words the standard deviation equals 0, if the error was always the same and gets higher if variations occur [32, p. 108 f.]. But as [33] pointing out, both methods in combination are not always the perfect tool to receive a good impression, when for instance on a small amount of samples, extreme outliers take a high influence on the result. Less sensitive for outliers is scoring with *percentiles*, since it shows the border below which point a certain amount of the data has scored. Primarily all results need to be ordered ascending and first then, the information can be extracted at which point a particular percentage of the data is situated [34].

Also for the *distance*, there are many implementations with different meanings for various purposes. Below, three exemplary variants are listed, to give an insight about the sense of this part, where a and b are two vectors with D dimensions (e.g. D = 2 for the

two dimensional image coordinates).

$$euclidean\ distance : d(a,b) = \sqrt{\sum_{j=1}^{D}(a_j - b_j)^2} \tag{4a}$$

$$canberra\ distance : d(a,b) = \sum_{j=1}^{D} \frac{|a_j - b_j|}{|a_j| + |b_j|} \tag{4b}$$

$$cosine\ distance : d(a,b) = 1 - \frac{\sum_{j=1}^{D} a_j \cdot b_j}{\sqrt{\sum_{j=1}^{D} a_j^2}\ \sqrt{\sum_{j=1}^{D} b_j^2}} \tag{4c}$$

Equation 4a represents the direct distance from a geometrical point of view and is probably the most easiest to imagine. In turn, 4b is a bit more complex, as it is more sensitive for measures distributed in the origin's vicinity. Whereas an approach from another perspective in 4c, the angular distance without the scale of the two vectors is compared. Another notable, yet simple variant is the *hamming distance* that displays the number of different entries from the two containers, which has because of its simplicity (more precisely ambiguity) a minor significance in numerical areas [35].

2.2.3.3 Feature space

A feature space, is a solution for the graphical representation when it comes to evaluating arbitrary features. It is designed as a special kind of a scatter plot, meaning when a particular amount of features are compared, each of them is marked on an own coordinate axis of the plot. Theoretically a feature space could handle infinite features, a limit is only given by the lack of displaying higher dimensions [36].

As an example in table 2 an artificial, sparse dataset without labels is given and its corresponding feature space is observable in fig. 9. From this point there are multiple ways to move on. One could start with clustering algorithms and find out how many groups are present or if it is just a homogeneous point cloud. And if such groups are retrieved further investigations could be initiated about determining the specific special meaning of them, whose information perhaps improve the further algorithm development. Or the features are solely observed to see, if and how much they correlate with each other, which is a sign that a feature adds no additional value to the dataset. This would be the case, if the points form a diagonal line. A good example would be variance and standard deviation, as both carry the same information but one is the squared result. Especially for classification, this utility can be used to analyze the current state of overfitting, which will be handled in chapter 2.3.4.

Table 2: An artificial dataset for a feature space.

Feature 1	Feature 2
6	3
5	2
0.5	2
5	1
1	3
2	4

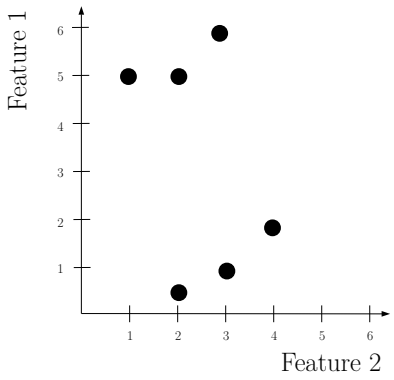


Figure 9: The feature space of table 2.

2.2.4 Validation of learning algorithms

The preceding sub chapter describes the error measurement for classification and regression. Their results usually stand for a very direct assessment of specific (not generic) circumstances. For instance, these error scores may be low, but it isn't clear if the parameters were already adjusted to the noise (i.e. overfitting) and thus fail on forthcoming unseen data. Then again, to retrieve a more robust representation of the algorithm properties a more general view on the problem is required. For this there are additional techniques, which are described below.

2.2.4.1 Cross-Validation

If the provided dataset consists of k samples to train a model, then this set is split up with one part to commonly continue with the training, where the other part simply is used for verification. The decision about how much data is used for training is not easily answered, as too less results in a poor adaption of the model whereas too less verification data lowers the score's validity. There are many different implementation and briefly explained in the following. In *holdout validation* usually $^2/_3$ of the data is used for training, it is less suitable for small datasets due to its single and thus biased estimate. In *k-fold cross validation* the split is random and creates k equally sized subsamples. As k-1 instances are used for training, where the remaining sample's intention is the validation. This procedure is repeated k-times to ensure each subset was used for validation once, the total score is calculated by the average of all subset results. Similar to this is the *Leave-one-out cross validation*, the difference is that *k-1* samples are used

for training so only one can be utilized for verification. *Jackknife* also just leaves out one sample like before but additionally per each fold further comparable values (such as *means*, *variances*) are calculated. The average of these statistic is then compared with the ones, when the algorithm was trained with the whole dataset to figure out a potential bias. Another option is *Bootstrap*, where a sample is randomly selected and but does not get removed from the selection space. In other words one sample can be chosen multiple times, where others won't get touched at all. The selected ones are used for training and the evaluation is processed with the rest [29, p. 539 ff.].

2.2.4.2 Hypothesis testing

Here a priori knowledge about the statistical distribution of the null and alternative hypothesis (H_0 and H_1 respectively - e.g.: is it class *Class A* or *Class B*) is necessary to minimize the probabilities of false decisions. Basically reject H_0 if $\Lambda(x) \leq \eta$, with the likelihood ratio $\Lambda(x) \equiv \frac{p(x|H_0)}{p(x|H_1)}$, where $p(x|H_0)$, $p(x|H_1)$ are the density functions of a sample x. There are plenty of methods in turn to define the threshold η, such as a *Bayesian Test* where simply right hypothesis with the higher probability is chosen: $\eta = \frac{\pi_1}{\pi_0}$, with $\pi_{0,1}$ as the prior probabilities of H_0 and H_1 respectively. If the parameters are biased the threshold is extend in *Risk-adjusted Bayesian test*: $\eta = \frac{\pi_1 C_{01}}{\pi_0 C_{10}}$ where $C_{10,01}$ are the weighs to level out the differences. [29, p. 541 ff.].

A special kind of this method is the significance testing, where H_0 is inspected alone. This for instance occurs, when in a scenario H_0 predominantly is chosen, or only the statistical distribution of H_0 is known, but not not from the alternative hypothesis. In other words this test states if a case is significant enough to reject H_0 which here would be the chosen one ordinarily [29, p. 542, 545 f.]

2.3 Decision trees

Ordinarily machine learning methods are delivered with many parameters that take influence on the result differently. Often they seem to be of an abstract nature, hardly to comprehend if one is not experienced in this area. For example the classifier from the scikit-learn library of a SVM comes with 14 different parameters [37]. In comparison to this, decision trees count to the fewer examples under these techniques that can be explained very vividly.

2.3.1 An example

In the following, a basic artificial constructed example shall introduce the topic.

Table 3, is the training dataset that is used for the task to separate fruits from each other. Starting very simple, a fruit could be separated by color. And because the dataset is then separated, further sub-datasets are generated, with less entities.

Table 3: Different properties of a fruit sample, for example 1

Label	**Freckled**	**Shape**	**Color**
Apple	no	round	red
Apple	yes	elongated	green
Apple	yes	round	yellow
Pear	yes	elongated	green
Pear	yes	elongated	yellow

Table 4: The sub datasets after it was split by the color-property

(a) dataset with red fruits

Label	**Freckled**	**Shape**	**Color**
Apple	no	round	red

(b) dataset with yellow fruits

Label	**Freckled**	**Shape**	**Color**
Apple	yes	round	yellow
Pear	yes	elongated	yellow

(c) dataset with green fruits

Label	**Freckled**	**Shape**	**Color**
Apple	yes	elongated	green
Pear	yes	elongated	green

In table 4 the generated subsets can be seen. According to this sparse dataset there is already a first result: all red fruits can be identified as an apple. The others need to be processed further. Where in *(b)* the fruits can luckily be separated by another split concerning their shape, it is not clear how to handle the ones in *(c)* where the properties are identical but with a different label. In this case there are two options: either there will be an *unidentified* classification or further properties have to be investigated.

This is a simplified way to construct a decision tree. The elaborated solution for this example can be seen in fig. 10. The question remains, what will happen if for instance a red pear appears. This is one reason why it is useful, if not even required to provide a huge dataset with as many different cases as possible.

2.3.2 Terminology and a brief overview

The probably most popular decision tree algorithm is called *ID3* [38], which is also used in this section to explain the behavior of these kind of ML-technique. Any tree is constructed as a connection of many *nodes*, whose amount are likely to increase non-linearly from layer to layer. In general there are only two types of nodes. An *internal node* queries a given sample and sets the continuative path along the algorithm, they occur from the initial stage until one step before the actual prediction. Internal nodes usually have a preceding super node and subsequent sub nodes, also known as *branches*. If a node has no subsequent branches it is called a *leave* (sometimes a *terminal* too).

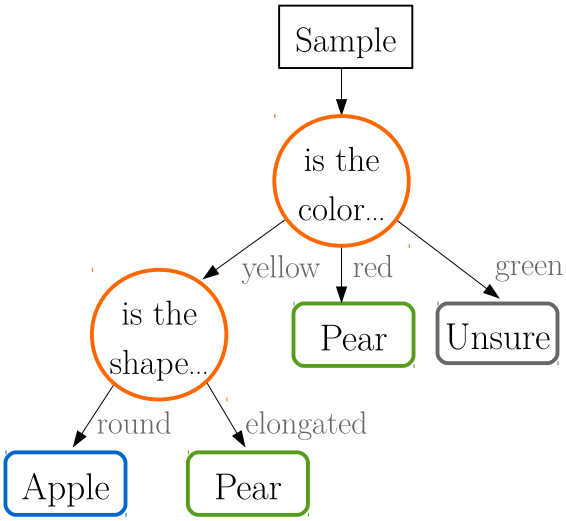


Figure 10: The decision tree as designed in the example.

However each tree starts with one initial (internal) node, known as *the root*. The *depth of a node* is the amount of decisions instances laying in between the root and the current one. Whenever a decision tree is splitting a node and forwards datasets into exactly two subsets it's called a *binary tree*, whereas trees with more than two branches are handled by the C 4.5 algorithm. When constructing a decision tree, either the full dataset is available from the very beginning, or is continually updated, which is called an *incremental tree design*. More precisely, either the existing classifier gets discarded and a new training from scratch with the updated dataset is required, or the current structure gets adjusted. [39], [40], [41].

2.3.3 The mechanisms

Decision trees are similar to the famous game *twenty questions*, where the a group of people have to ask arbitrary questions, answerable with either yes or no (e.g. „is it a meal?“ or „is it big?“). The goal is to identify the unknown object/person/etc. with as few questions as possible.

The machine learning technique based on this approach tries also to identify a given case as quick as possible, with the help of concise queries. For this, a dataset with as many possible occurring states needs to be provided, which is used to train the decision tree (see chapter 2.2.1). When the dataset is retrieved, the next step is to begin the build process by starting to split on one of the given features. In a binary tree, a split generates two sub datasets who together resemble the same information, that could be extracted from the original one. The subsequent iteration continues the splitting with the sub-datasets. This procedure is repeated until the design of the constructed tree structure fits the specifications (e.g. full dataset splitting, or finishing earlier to avoid

overfitting).

However the question remains where to set a split. Often in machine learning, a method is granted more time for its training phase, as long as the processing speed in the testing phase (i.e.: the actual application) increases. For achieving this goal, either the feature extraction is accelerated, or on the classifier's side, the amount of queries/repetitions until the actual prediction is kept low. The later minimization is often implemented with the *information gain* [42], it is shown in equation 5. With A as the attribute that splits the set S and H as a metric function.

$$I(S, A) = H(S) - \sum_{\theta \in A} \frac{|S_\theta|}{|S|} \cdot H(S_\theta) \tag{5}$$

There are plenty options for the metric H, where the most common ones are presented subsequently.

Gini-Index

This measure is commonly used to split the parent's node dataset, if the selected feature is of a categorical nature (e.g.: music genres) [43]. It is shown in equation 6 with p_c as the relative class frequency of a certain instance of C classes.

$$H(S) = 1 - \sum_{c=1}^{C} (p_c)^2 \tag{6}$$

Entropy

Very common next to Gini is the Entropy measure, shown in equation 7. According to Elkan's opinion in [44] different criteria do not have that much effect on the outcome, because of their similarity. However [45] states that it rather choses smaller counts but many distinct values.

$$H(S) = -\sum_{c=1}^{C} log_2(p_c) \cdot p_c \tag{7}$$

ROC-based splitting

To mention an alternative, that is independent from the information gain, Ferri at al. suggest in [46] to chose the attribute that retrieves the highest local area under the curve (see chapter 2.2.3.1).

Further options to increase predicting performance

A notable variation in this area for improving the prediction accuracy in the testing phase is *Bagging / Bootstrap Aggregating*, where per classifier a few random samples of a given set are chosen for the training (this includes cases where some data can be selected multiple times, but others not even once). Alternatively in *Boosting* multiple

weak learners (which are more or less accurate than a random guess) are combined to complement each others weaknesses, for instance on a specific feature [47, p. 23 f. p. 47 ff.].

A complete different approach is to avoid carefully selecting the best split and replace it by a random separation [48]. This might yield a processing speed acceleration during the training phase, as the overhead of the metric's calculation falls away, when even though it is especially for very inhomogeneous datasets possible to generate an unnecessarily complex tree design. Particularly when a method is applied on a mobile platform or a robot, not only time is a criteria but the energy consumption carries valuable consideration too, since less device accesses consumes less energy compared to long-term calculations. Unfortunately the drawback of this non-optimized tree layout is, that might demand more time during its actual prediction in the testing phase. This kind of design is unsuitable for the problem handled by the thesis, but it might come in handy on cases with small datasets whose worst layout can't become a too complex structure.

In conclusion the decision tree developer has to design a layout specifically according to the application. Including the considerations about the type of estimation (numeric or deterministic). Furthermore if it is required to perform as a real-time application, or if it is used in decision making (e.g. evaluating a dataset in medical consultations). Also choices if the accuracy is the most important attribute, or if the priority is to keep the processing speed below a certain limit, need to be reasonable compromised. Besides these trade-offs it is recommended to provide an uncorrelated feature extraction as origin of the algorithm's prediction quality.

2.3.4 A decision tree in feature space

In this section the mechanisms of such a classifier is visualized, to outline its attributes and mechanisms. Also to point out the negative aspects, in order to show why a decision tree alone is not enough and a RF is preferred, as it eliminates the weaknesses.

When a decision tree was trained to separate all the entities for a given dataset, then it is likely that an overfitting has been achieved, which leads to misinterpreting the actual testing data. This problem shall be explained with an artificially constructed example below.

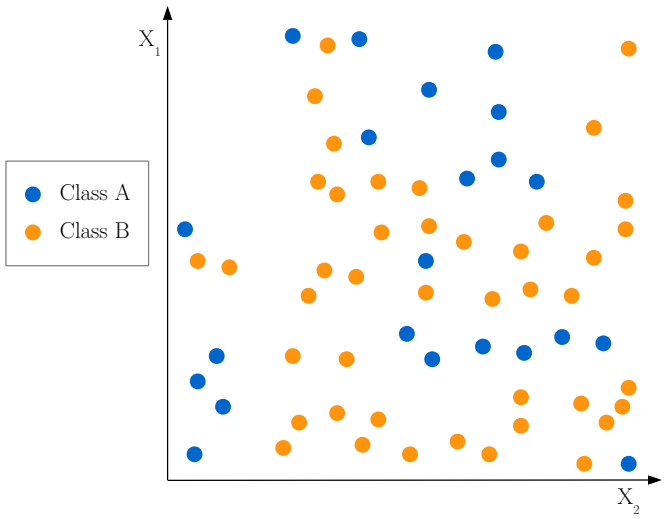
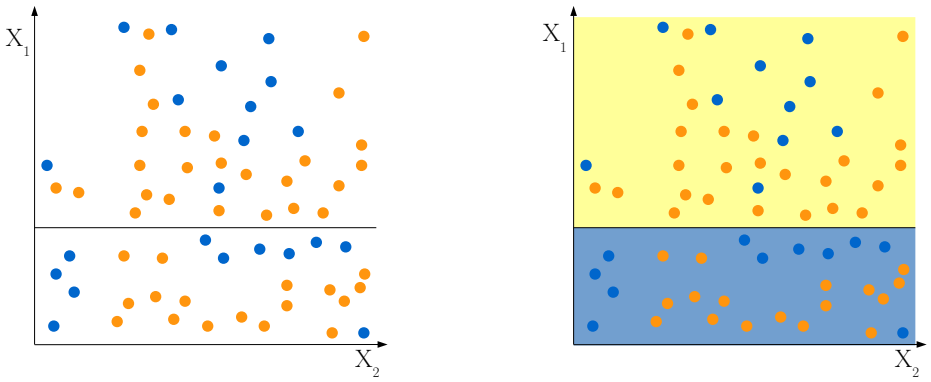


Figure 11: A feature space of random values to demonstrate the meaning of a split.

In this example a tree is trained with a dataset concerning only two features: X_1 and X_2 to be seen in fig. 11. For this machine learning technique a split can either be a vertical or a horizontal line in feature space (concerning X_1, or X_2 respectively). The result of a first separation can be observed in fig. 12 a). If one forces the training process to stop at this position, the testing data then will be poorly separated as demonstrated in b) with the yellow area as *class A* and the blue area for *class B* prediction. It's clear to see that this isn't very accurate, observing the inhomogeneous distribution of class samples inside both fields.



(a) first split

(b) current tree prediction

Figure 12: The feature space after the first step of building a decision tree with a given dataset.

Continuing and fully splitting the given training dataset generates two special cases one needs to be aware of, they are marked as gray areas in fig. 13. For the lower right spot it could be just an error in the training data set or just a correct but special case. Same situation applies for the spot in the middle, but here one could believe there was just a typo when building up a dataset and it actually belongs to the blue group below. Assumed these were mistakes and should actually be part of the *class B* (yellow) area, if the data however is then strictly separated by the given (noisy) dataset, future predictions in these areas will be false, this is called *overfitting*. The opposite phenomenon *underfitting* can be observed in the big gray area up-left in the graph, where one cannot be very sure if all samples in this area can be identified as *class A* (blue) because of the missing training data for these cases.

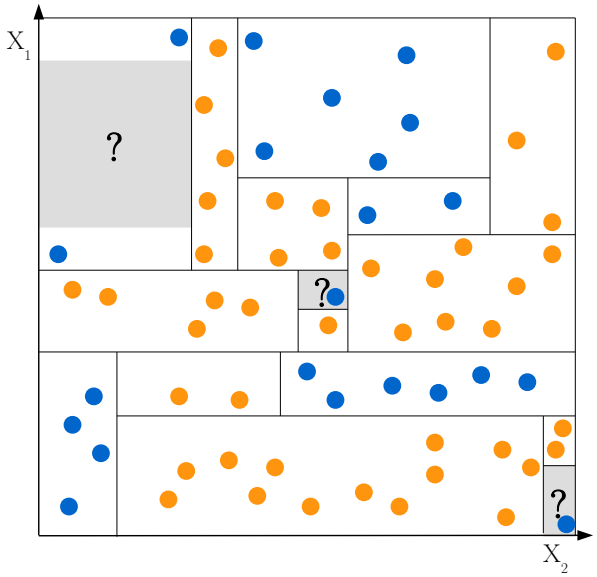
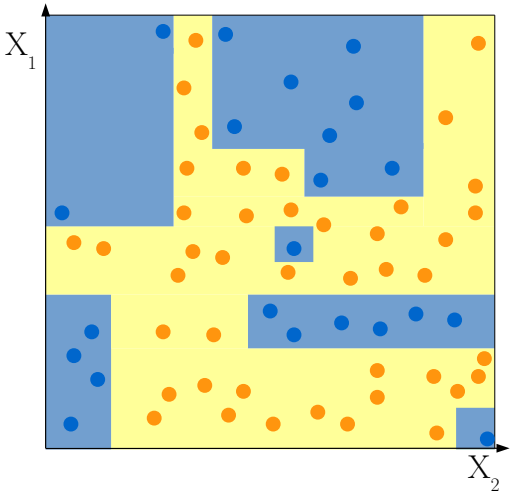


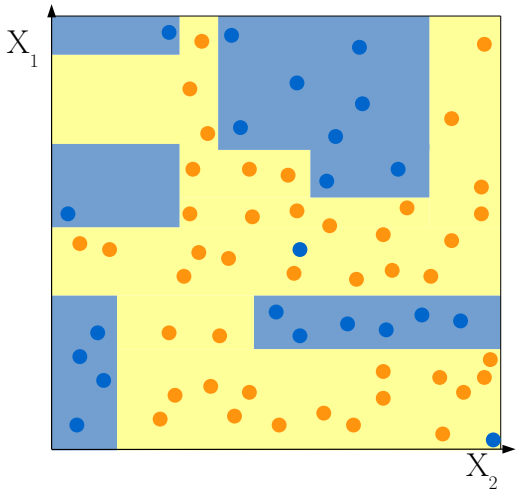
Figure 13: The feature space after full separation of all known entities.

The results of decision tree training with a insufficient dataset, that leads to over- and underfitting for this example can be observed in fig. 14. Subgraph a) greedily separates the data as best as it could, which leads to wrong labeling areas, compared to the ideal result in b).

Certainly this is a behavior that requires further attention to avoid false predictions. On possibility is to *prune* the tree, which means to prevent a complete separation of the dataset entities, as shown above. Either this is implemented during the training phase (*pre-pruning*) or after the tree was built (*post-pruning*).



(a) real separation



(b) ideal separation

Figure 14: Comparison of the final result of the training phase between a real and ideal sample separation.

That implies the question about when or where to snip the tree. For this, additional methods are necessary that gauge the current state of overfitting. Selected tools for this purpose are briefly mentioned in the following. *Reduced error pruning*, where internal nodes are replaced by a leave (prediction) and evaluated if the error is equal or smaller than the previous state, which then should replace the current subtree with the leave. *Cost-complexity pruning* works on a fully trained tree, where bottom-up iteratively branches are replaced with leaves. This is repeated until the main tree is only a

leave too. Under all solutions during these iterations, the error rate is compared and the best case is chosen. In *critical value pruning* each node is checked, whether it exceeds a critical value and if not it will be pruned with an exception: the branch is kept if its sub-branches don't meet the pruning condition, because then they contain relevant splitting information. The higher the critical error was chosen, the smaller the resulting tree will become [49].

2.3.5 Conclusion

Summarizing the decision tree algorithms with its strengths and limitations [50, p. 45 ff], [51], [52]:

Positive aspects:

- Performing well with a high accuracy even in image processing
- Easy to visualize and demonstrate.
- Works with numerical and deterministic data.
- Performs well on reasonable amounts of computing power, even on a large dataset.
- Have various applications in medical decision making.

Negative aspects:

- Might generate overly complex models, depending on the training dataset.
- A good design demands a significant, quality dataset, which is not always available in a medical area.
- Sensitive to overfitting.
- Is forced to process a parent node first, whose decision might not be defined for all cases.

A decision tree's solid ability to minimize the amount of queries before a prediction, makes it a favorable candidate for real-time calculation demands. Also the high accuracy even in image processing, supports the choice of being an appropriate tool for the ultrasonic tracking. But it cannot be ignored, that the training dataset will be small (i.e.: it needs to be generated by the algorithm itself based on a few information, thus being sparse), it must then be taking into account that it might not cover all possible states in its training phase. The outcome of learning a tree in such a manner, will surely be burdened with overfitting and the fact that the tree needs to process parent nodes, who were trained with insufficient data, that confuse the classifier on unprepared situations.

To reduce the negative but obtaining the positive aspects, a general variation of working with decision trees was provided provided by L. Breiman as he introduced in [53] the RF.

2.4 Random forest

This ML-technique ranks amongst others in a group of algorithms that consist of multiple classifiers that aggregate their individual results and collectively submit a single prediction, which is the sole output used for further processing afterwards. As there are many variants, referring publications are usually collected by the name *ensemble learning* or *combined predictors*. Simply spoken all of such methods run many instances of an underlying fundamental algorithm, each of of them with a slight variation. This difference however is the point were one could further split up these methods. In one group the training is designed in a kind of manner that every classifier is constructed independently from each other, resulting in an appropriate low error rate already per classifier. the other group in turn, generates its individual elements on a shared basis [54]. Exemplary methods are *Bagging* and *Boosting* as mentioned in chapter 2.3.3.

Starting with a decision tree, it is rather easy to move on towards ensemble learning. Initially, multiple trees need to be generated and form together a decision forest. If this forest is taken as a whole algorithm, a sample prediction can be estimated simply by voting. This is graphically summarized in figure 15, where circles constitute a decision, rectangles a prediction, an orange line as the way of the same input sample for each classifier and the blue entities as illustration of the individual decision path inside each tree. Already by this figure a few properties are observable. The left algorithm visibly needs less resources in memory. Also its processing time here is ≈ 5 times faster compared to the forest (consisting of 5 individual trees and thus the 5 times decreased speed), if the trees aren't parallelized. However, the left algorithm doesn't seem to be trained very well, or at least the input sample is hard to safely predict, which leads to retrieving an already wrong prediction that is used for further processing. Also a few trees inside the ensemble have false estimate, but the actual algorithm output is given by a subsequent voting procedure of all individuals and thus being able *averaging* errors simply. For the voting system many options are imaginable, such as the majority vote, where the prediction is chosen that was selected the most amongst all other entities. Contrary to this example, if the ensemble is not interpreted as a binary classifier (i.e.: having more than two possible options to predict), a median vote might be a suitable option as well.

Many different ways for constructing the single classifiers in the ensemble are conceivable and because it is a comparably rather old procedure, there are different publications suggesting methods to efficiently arrange the tree design.. In connection to the methods shown before Schapire et al. evaluate the effectiveness of utilizing *Boosting* on this purpose [55]. *Bagging* in this area is investigated by Breiman [56] with the statement, that this method already might cause substantial gains in accuracy. A few more notable investigations are mentioned in the following. Dietterich inspects randomizing the split selection from a collection of n favorable splits, to enhance the outcome of the results [57]. Ho states, that adding more trees might help to overcome a poor generalization, where likewise the accuracy increases monotonically by the amount of trees being part

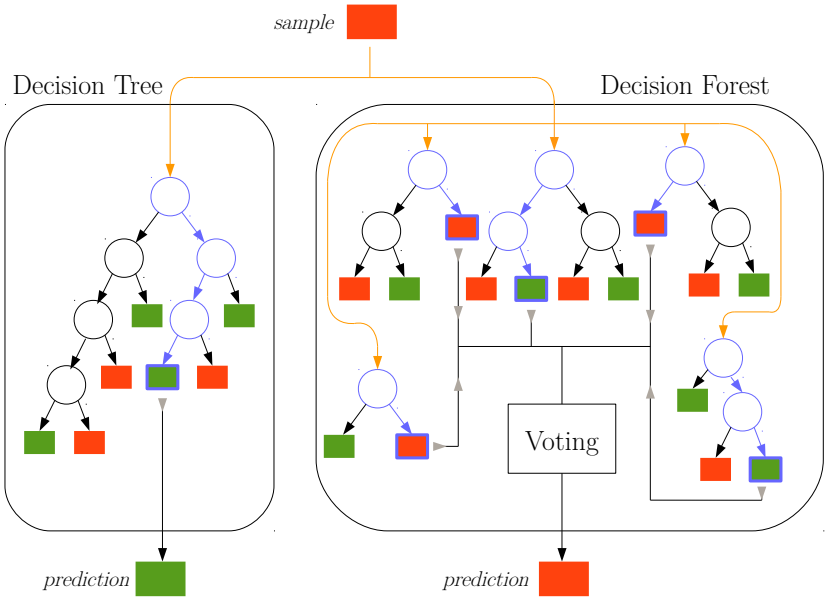


Figure 15: A graphical comparison between the mechanics of a single decision tree and a forest.

of the ensemble. Also different feature vectors, number of subspace dimensions or tree growing algorithms don't seem influence the classification accuracy much [58].

With this introduction, the mechanism of a random forest can be explained. Basically it continues the idea of Bagging by increasing its randomness, with the aim of having an ability of a better handling the unseen data. Instead of figuring out, what would be the best information to chose (e.g. preselecting the significant features of the dataset for each tree), this shall rather be done less precisely by working with a unoptimized information basis. It appears obvious when a training set doesn't cover all occurring cases of the testing phase, that it is needless to perfectly adjust to the complete training set, that decreases the generalization capabilities (a possible outcome is described in feature space analysis of chapter 2.3.4 or figure 7). More precisely every single tree is generated independently from its adjacent classifier-relatives. Further on every tree-node is selected as the prime split amongst its random (thus the name) subset of features, but with the same distribution. Especially to demonstrate the generalization abilities, there is a very useful measurement tool, which is the so called *out-of-bag error*. It is the mean error if a sample was evaluated by all trees that didn't use it during their training phase [53].

A random forest is already built in a few steps. Starting by creating n_{Trees} bootstrap samples, where each tree shall grow unpruned (see chapter 2.3.4) but instead of using

all of the provided features, an arbitrary choice of the whole collection is used for a current node. Now for predicting samples all results of the n_{Trees} classifiers need to be aggregated, for instance by majority vote or calculating a mean value (preferably for regression tasks) and hence deliver the single result [59].

In [53, p. 7] Breiman attributes following properties, which are remarkable to these procedures. Their accuracy is comparably good or even better than the AdaBoost competitor, while being quite robust in terms of handling data with outliers and noise. It is working with a faster processing speed as against Boosting or Bagging, also comes in modest and is easy to parallelize (thus represents itself interesting for rapid or even real-time applications). Its structure hands in useful information about estimates of error, correlation, variable importance and strength. These information are important for feature engineering or the algorithm developer at all. It's fair enough to cite the handbook of random forests [60], which interferes with the mentioned attributes but also continues with the following statements. When talking about accuracy in classification the random forest shall be as good as SVM, with its technique that maintains its accuracy even if around 80% of the actual necessary data is missing and also possibly because of it's attribute of harmonizing the error of an unbalanced dataset class distribution. Then in turn Liaw et al. [59] state that the effectiveness does depend on various parameters. Amongst others issues, having an highly unbalanced class distribution may recognizably lead to a false prediction due to the voting system, since the commonly used majority vote may be inappropriate at this special case. As the algorithm is perfectly parallelizable it might come in handy to run many random forests on different machines and combine each result together to calculate the final outcome. The amount of involved features doesn't take that much influence on the results, since already only one can be sufficient for some data. However the advise was given to rather increase the amount of features, when there are generally a lot of features where only a few are determinative. Breiman in turn suggested in [60] a more empiric approach of using twice, half or just the amount of features from the default case and pick the one with most promising results.

2.5 Discussion

Tracking on medical ultrasonic image data comes with many influence taking factors, that carefully need to be weighted to develop the optimal algorithm. In comparison, following a ball in a two- or even three-dimensional scene is rather easy due to its almost static appearance (a sphere with the same size on each frame), subjected to physical laws that help to predict at least a rough position on forthcoming fames, if not interfered on the way. Balls are often robustly identified with circle recognition tools like the hough-transformation [61], or if the scene is even less complex and stays static during the whole tracking period, a simple image-subtraction might already be enough to identify the moving object. However, observing an object and its scene on ultrasonic image data lead to the impression, that these kind of problems are substantially more difficult and might not be able to be handle by rather simple approaches at all (see fig. 5 in 2.1.2 as impression). As for instance it is perfectly normal that the appearances of scene and ob-

ject change dynamically over time, occurring occlusions of relevant and irrelevant parts and also the object's dimensions may easily change. Consequentially a more perceptive method is needed, capable of such factors. Often machine learning methods are used on similar purposes, such as face or handwriting recognition and the like. Especially because of these two examples on object recognition with a lot of possible variation in the looks (e.g.: distinguishing an arbitrary letter under the influence of different style of writing, cursive or capital format, etc.) and their problem's similarity to the ultrasonic tracking issues, as also successful ML-submissions to the CLUST challenge, it appears promising to assemble these potentials in the own tracking algorithm.

Deciding to use ML leaves up the question which of all available algorithms is suitable for this situation. Because of such a massive amount of choices (e.g. see chapter 2.2.2, or the well known overview map of scikit-learn [62]), the right option needs to be encircled with the help of appropriate queries. It's fair enough to initially decide between classification, regression or clustering. Since it is already known what shall be inspected, ambiguous techniques, like the ones from clustering, are probably a lot less efficient because there is already a clear border between the possible occurring groups. Further on, a regression might only be worth considering when the trajectory shall be learned (more precisely the particular *continuous* image coordinates on the image's y- and x-axis). Leaving up a majority weight on classification procedures, being capable of classifying either positions or pixels affiliation for instance. Even though, most of the algorithms are often capable for both options. Having a closer look on a given ROI. Because many pixels of its pattern already change from one frame to another, the classifier then is probably going to handle a lot of unseen cases or situations and still needs to give a solid prediction, thus a well generalization is demanded. Further on, if the algorithm should be used right away after the training input was given, a rapid training phase is necessary. This excludes neural networks,because of their training may last hours (e.g. the variant presented in [9], took around 1.5 h for adjusting a CNN to the CLUST Data). The next commonly used classification method is the SVM because of their accuracy. But also these might take in multiple hours of time for preparation, as inspected for face recognition in [63]. A lesser known method is the *discriminant analysis* and according to a multi-class experiment in [64] is quite competitive to other classifiers like SVM. Unfortunately [65] indicates that this variant tends to overfitting that probably impedes handling the various unseen ROI-appearances during tracking. Sollich and Krogh [66] are discussing overfitting and realistic ensemble sizes, it seems adequate to state ensemble methods hold a higher potential to avoid overfitting because of their averaging technique, if it was not over-optimized (e.g. when the individuals in a decision forest are adjusted to the whole dataset. Then the result is not likely to change after the averaging, since all classifiers would be alike - see chapter 2.4). Which minimizes a lot more the amount of appropriate ML-techniques to these kind of composed classifiers.

In general RF as an ensemble classifier is known to handle unseen data well, that suits the requirement of the ROI appearance. With short training and testing phases it's

applicable in reality with no further extensions on one hand (e.g. it is not required a session with the patient for training the algorithm and a second session for the actual application), as on the other it greatly reduces the evaluation time during development due to the lack of long waiting periods until the algorithm is finished and ready to use. However, there are quite contrary meanings about the accuracy. Douglas is comparing the accuracy of K*, Naive Bayes, SVM, Decision Tree, AdaBoost and RF in [67], for a 10-fold cross validation on MRI image data to guess mental processes. The RF has achieved maximum accuracy with 92 % followed by AdaBoost's 91% whereas the worst result came from the SVM with 84 %. This impression is also supported by the results of [68], where inter alia RF, SVM, neural networks and linear regression are compared with each other on 11 different datasets (each separated further in training and test sets) from the UCI repository. Whereas in [69], RF has the highest FP-rate (segregating non-phising E-Mails) but still the lowest FN-rate (actual spam that would be let through) in predicting phising E-Mails, which still is considered as useful, when it is outperforming the other classifiers e.g.: neural networks, SVM or linear regression. Piater et al. [70] investigate the classifier's performances on image data. Where for the MNIST dataset (handwriting) RF achieves a position in the midfield (3% error rate) against a winning implementation of a one layer neural network, followed by the SVM. Also an average performance for the COIL-100 dataset (3D objects) for the RF (1.17% error rate) as against a better result of a linear SVM (0.4% error rate). For the ORL dataset (faces) in turn, the performance increased again, RF takes with SVM a leading position (both around 1.24% error rate). As in the last dataset OUTEX (textures) RF has a incredibly high error rate of 66.9% but still better than a linear SVM with 71.99%, or a classical decision tree with 89.35%, even another ensemble method, the extra trees, consisting of 1000 classifiers only achieved 65.05%. However [71] clearly states SVM as the winner of the direct comparison with RF, in almost all 22 datasets about a deterministic cancer diagnosis.

After all, these individual results cannot be put in contrast directly with each other, but it is meant to leave an impression, for supporting the choice of the right method. Besides the different winners on each test may certainly be influenced of the particular test setups as for instance the specific parameter settings, as also by the type of each datasets (image analysis, cancer diagnosis, etc.) that makes it hard to clearly settle down for the best decision, on the other hand this leaves up room for further investigations for variating the thesis procedures with different ML-techniques. Even though RF cannot be seen as the only algorithm-of-choice, it generally verifies a relative high prediction accuracy, adding a relative quick training/testing phase and a well reduction of overfitting, that makes it a excellent choice for the tracking algorithm. To mention published algorithms, Criminsi et al. describe in [72, p.193-209] an algorithm for anatomy detection implemented with regression forests and in [72, p.247-260] classification forests for segmentation of sclerosis lesions. The involved CT- or MR-images are not quite the same as ultrasonic data, but still they are similar, which makes this decision about utilizing RF more confident.

As there are already plenty of RF implementations, tested, evaluated and updated accordingly in many different cases, it seems more than appropriate to pick one of these existing solutions too. This positively helps to concentrate on the actual tracking problem. Additionally, by being inspected from a larger group of people the implementation is very likely to be at a highly optimized state. Also a big supporting community that usually comes along with such programs, are arguments that may greatly decrease the developing time. As a drawback, the user of such utilities often is limited to the specific implementation (because of proprietary software, or too complex coding that would claim to much time for reengineering) that may or may not be sufficient for a current approach, which excludes applying already small variations on their base algorithm. In [73] multiple implementations of RF are compared with each other on a numeric, airline-based dataset and summarized below. The properties of the implementation from

- *R* are slow and memory-inefficient with an average accuracy.
- *scikit-learn* are faster, more memory efficient and a slightly better accuracy.
- *H2O* are alike with the ones of scikit-learn, but with an even higher accuracy.
- *Spark* are slow and a lower accuracy.

On another test for MNIST (handwriting) and for a dataset about forest covertypes from the UCI repository, claims that the scikit-learn variant has a minimal higher accuracy than H2O [74].

At this very initial stage of development a fully optimized design is not in focus, whereas a lightweight and swift development environment (such as programming with Python) is a lot more helpful for constructing first prototypes, where ideas with a comparably low effort are implementable. Since H2O is a Java library, the decision falls on scikit learn as it is free, apparently working very well and perfectly able to be used within Python programs.

3

Algorithm development

Besides the already wide variety of proposed solutions by the CLUST challenge participants (e.g. chapter 1.2), there are further options to handle the tracking scenarios, a different kind of way. This section presents the own developed approaches, where at first a solution concerning about the ROI-appearance and secondly a solution that focuses on global features, is presented.

The *pixel classification* describes an approach, where the underlying RF decides if an arbitrary sample pixel is part of the ROI and thus distinguishing the *object* from its *environment* bit by bit. The classifier of the later method predicts the object's position on a learned trajectory according to pattern-variations, which are extracted from a static image section in *landmark estimation*.

Both algorithms make use of the correlation coefficient. It is an algorithm for locating a reference patch F_r in a corresponding image F_s according to similar pixel intensities. If the image dimension of F_r is a lot smaller than F_s, then the most similar position can be detected, by moving the smaller F_r upon the larger F_s (i.e.: a sliding window). On each position a different similarity is calculated and a result matrix is generated, so in the end the coordinates of the position with the best result is likely to be the new center of F_r in F_s. An example is given in figure 16, where $\mathrm{r}(\Delta y, \Delta x)$ is the chosen estimate. Formula 8 shows the mathematics.[75, p. 395 f].

$$r(\Delta y, \Delta x) = \frac{\sum_y \sum_x (F_s(y+\Delta y,\ x+\Delta x) - \bar{F}_s)(F_r(y,\ x) - \bar{F}_r)}{\sqrt{\sum_y \sum_x (F_s(y+\Delta y,\ x+\Delta x) - \bar{F}_s)^2 \cdot \sum_y \sum_x (F_r(y,\ x) - \bar{F}_r)^2}} \tag{8}$$

r(Δy-2,Δx-2) = 0.757	r(Δy-2,Δx-1) = 0.763	r(Δy-2,Δx) = 0.776	r(Δy-2,Δx+1) = 0.754
r(Δy-1,Δx-2) = 0.782	r(Δy-1,Δx-1) = 0.794	r(Δy-1,Δx) = 0.801	r(Δy-1,Δx+1) = 0.792
r(Δy,Δx-2) = 0.782	r(Δy,Δx-1) = 0.842	r(Δy,Δx) = **0.882**	r(Δy,Δx+1) = 0.832
r(Δy+1,Δx-2) = 0.791	r(Δy+1,Δx-1) = 0.807	r(Δy+1,Δx) = 0.821	r(Δy+1,Δx+1) = 0.814

Figure 16: Example for the result matrix of the correlation coefficient.

3.1 Pixel classification

The idea for the following approach is to predict each part of the image whether it is part of the object or not. Since the smallest part of a digital image is a pixel, the classifier is trained to distinguish between the two classes (*object* and *environment*) on each of these elements. If these predictions are not parallelized it might demand a lot of time processing an image, but on the other hand the highest resolution is retrieved, assuming that the RF doesn't classify falsely. In the end an image of the estimated object will be generated, from where the object's center coordinates are extractable.

3.1.1 Pixel label generation

A sparse a priori knowledge was given with the ultrasonic images, providing the position of a ROI for just a few frames. Moreover unfortunately this doesn't contain any labels per pixel.

It must be considered that not all parts on the ultrasonic record can be clearly labeled, since for most of the sequences multiple objects are analyzed at the same frame. When there are for instance two ROIs on each frame, a decision about the label design has to be made. For instance all pixel for both objects could be labeled the same, as *object*, since both look related compared to the surrounding areas. Then this would greatly increase the amount of pixels for the dataset identified as *object* and additionally reduce the amount of same feature vectors but with different labels, when one feature combination was extracted from both ROIs (since otherwise, there would be one labeled as *environment* and the other as *object*). But then there is the drawback that an area could be falsely identified as *object* in the testing phase, just because of the labels in the training dataset. This occurrence usually happens only, if the two ROI are situated close together, since if they are far away from each other, the classifier's scope could easily be restricted, which excludes the false candidate from the very start. It probably might then be better, having only the pixel of the ROI be labeled as object and all others as another class. Then again with the drawback, that multiple same feature vectors could occur but with different labels, that lowers the amount of training data especially for the object.

The difficulties are, that there is practically no dedicated information that distinguishes the object from its vicinity and taking into account when it comes to apply the method in a real-life scenario, the user shall be discharged of as many settings as possible (i.e.: it is desirable, to only provide the initial position and the size of a tracking object as algorithm input parameters). Having this situation and because RF require a dataset, the algorithm necessarily needs to arrange one on its own.

A naïve. yet effective approach is to analyze, not the whole image, but just the area around the object. Exploiting the user-input of the position as virtual center of a ROI, a decision value then can be made from the small image section around the object, di-

viding the pixels in two classes. This threshold t_O is simply calculated by a mean value in the area, as shown in equation 9, with F_t as the image and $F_t(i,j)$ as the intensity value at a particular position, x and y as the center coordinates of the ROI retrieved from the user input.

$$t_O = \frac{1}{9} \cdot \sum_{i=y-1}^{y+1} \sum_{j=x-1}^{x+1} F_t(i,j) \tag{9}$$

In general, the so generated threshold is somewhere in between of the image data type's upper and lower limits. With this value, for separating the object from its environment by just a single boundary, would leave open only one option: splitting with a greater/smaller decision. But this separation wouldn't be appropriate, when for instance lower values than the threshold probably are part of the vicinity and higher intensities could be a margin from the object. Because it is the center of the ROI intensities distribution, which means there are pixel on the upper and lower side of the t_O boundary, which den are isolated incorrectly. It would still be inappropriate, if only those values are not labeled as *environment*, who are equal to the threshold, since the intensities of the ROI vary slightly around the threshold, or as a worst case wouldn't retrieve any result at all. So it is clear to see, introducing a tolerance t_t ranging around the decision value could solve the problem, as described in equation 10 below. A sample (i.e.: pixel) then is identified as one of the two cases when:

$$label\Big(I(i,j)\Big) = \begin{cases} object & , t_O - t_t \leq \ I(i,j) \ \leq t_O + t_t \\ environment & , otherwise \end{cases} \tag{10}$$

When applying this basic kind of identification to a ROI, mostly it is able to divide the two groups (*object*, *environment*) visually rather good from each other. For instance, the sample shown in fig. 3 e) could easily retrieve a sufficient separation, since the difference of the intensity-average between both groups is big. However one cannot exclude cases of pixels being in the range, that would be identified as *object*, but are actually noise in the vicinity, which match the requirements just by random. Even worse, when it comes to apply the method on the ROI as shown in fig. 3 g), where the *object* is only separated by its margin from the *environment*. Here, most of the pixels would be identified as *object*, except of the ones with higher intensities from the objects margin. This is because both groups have more or less the same average intensity level and thus being identified as *object*, according to the rule of equation 10. Therefore, further mechanics need to be included to prevent this kind of misinterpretation.

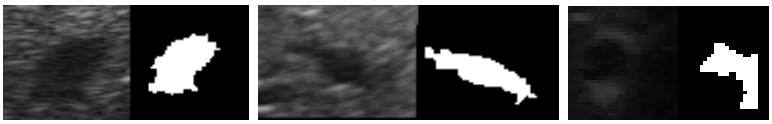
For the algorithm, besides the position and size, an additional information about the object can be achieved by practically no further efforts at all. Simply, by instructing the user to define the size of the image section in such a manner, that a constant fraction of the image size needs to be part of the object. For instance it could be stated that $n\%$ of the section has to be filled with the object as the remaining $100\% - n\%$ are part of

its surroundings ($n \in \mathbb{N} | 0 < n < 100$).

With this information it becomes a lot easier to prevent the errors mentioned above. Pixel around the image margin most probably won't belong to the object, whereas the ones from the center do. It is then adequate to start separating, where it is save to say an area is part of the ROI. Nevertheless a rule needs to be defined about which pixel should be analyzed first, which to processed next and finally when to stop the binarization. A common technique is to start at a certain (seed) point, remembering all adjacent pixel positions. When the current spot's identity is defined as *object* by the method mentioned above, it is then granted to recover all saved adjacent pixel positions before, otherwise not. This is repeated as long as there are no remaining pixels left to check, or the amount of ROI-pixels were reached (i.e. n % of all pixels). This method is commonly known by the name *flood fill*, *seed fill* or *region growing*.

Finally the separation method is assembled in three steps. In the first, the user's information about the ROI is retrieved (i.e.: position and size, with the precondition to define the size of the image section relative to the amount of the *object* pixels). Secondly, the decision value t_O is extracted as shown in equation 9, which is extended with a tolerance band to distinguish the two groups, as described with equation 10 and ultimately applying this kind of flood fill on the image section.

A few samples of this approach are exposed in figure 17, a) as a rather good example, b) with a moderate result and c) with the problematic background. The outcomes are visualized as binary image, where white pixels indicate the *object* and black intensities the *environment*. The results carry a good recognition value compared to the original ROIs next to them. Especially with the problematic tissue c), the advances of this method stand out. Since, the object's margin was not labeled incorrectly and where the back- and foreground do not differ much, the algorithm stopped adding *object* pixels, when their specific amount is reached. This retrieves a decent separation under various difficulties.



(a) MED02-3, ID 1 (b) ICR01, ID2 (c) ETH01-1,ID 2

Figure 17: Results of the artificial label generation.

At last the so generated labels need to be combined with the appropriate feature vectors. The next paragraph discusses the feature extraction and how they are linked together

for a specific pixel.

3.1.2 Feature extraction

As illustrated in chapter 2.2.1, features can have a variety of appearances (such as a binary or multiple deterministic states, or even a continues range), leaving a large freedom for the design. What really matters for their value, is to remain virtually constant during all possible states, in a sense, that a well recognition of the involved classes along the object's trajectory can be guaranteed. Figure 10 initially showed, that the strictness of being forced to ideally stay correct in every case, can be eased by involving additional features, which combined are capable to figure out the right estimate of a given sample. But if a feature response is always distinctively different whenever similar samples are handled, it indicates a random behavior (e.g. noise). If such data is included to the training set, it doesn't positively influence much the later estimates, since it is only relevant for this special case but lacks of generalization, that provides little information to effectively separate the dataset.

The evaluated features were:

- Boxfilter responses based on mean value.
- Boxfilter responses based on local minimum.
- Boxfilter responses based on local maximum.
- Intensity stretching filter responses.
- Unprocessed pixel intensities.
- Responses of a correlation coefficient.
- Combination of the options mentioned above.

A boxfilter, sometimes called moving window, is a mask whose size is a fraction of the full image's dimension. It's moving stepwise over the whole image and applies for each position an attached filter-function, that changes its response value according to the currently involved pixels inside its mask. This means one gets different feature values by simply changing the mask size or the filter function. For the following features, this technique was implemented in different ways.

3.1.2.1 Mean-Boxfilter

Simply all intensities inside the mask are summed up and divided by the amount of involved elements, the result is applied to the center pixel (algorithm 1)

Algorithm 1: Feature extraction: Mean-Boxfilter

```
1  function meanInArea (img, filtersize F_fs)
   Output: feature image F_f
2
3  F_f ← copy(img)
4  for y_cur ← F_fs.y · 1/2 to img.y − F_fs.y · 1/2 inc +1 do
5      for x_cur ← F_fs.x · 1/2 to img.x − F_fs.x · 1/2 inc +1 do
6
7          curMean ← 0
8          for y_filt ← y_cur − F_fs.y · 1/2 to y_cur + F_fs.y · 1/2 inc +1 do
9              for x_filt ← x_cur − F_fs.x · 1/2 to x_cur + F_fs.x · 1/2 inc +1 do
10                 curMean ← curMean + img(y_filt, x_filt).intensity
11             end
12         end
13         F_f(y_cur, x_cur) ← curMean / (F_fs.x · F_fs.y)
14     end
15 end
```

3.1.2.2 Minimum/Maximum-Boxfilter

For this case the boxfilter was applied differently. The filter response isn't just applied for the center pixel of the mask, this response rather is assigned to all involved elements. Here the filter function simply retrieves the minimum or maximum intensity level inside the mask. Algorithm 2 shows the procedure (to avoid redundancy both filter were combined into one algorithm for the documentation), with *option* as switch to either find the minimum or maximum and the *curWindow* as the current filter section, extracted from *img*. To prevent the algorithm from overwriting already existing assignments, the mask needs to have a bigger step sizes than iterating every pixel, so only unprocessed data is handled (i.e.: step size is equivalent to the related filter dimension).

Algorithm 2: Feature extraction: Min/Max-Boxfilter.

```
1  function extremaInArea (img, filtersize F_fs, option)
   Output: feature image F_f
2
3  F_f ← copy(img)
4  for y_cur ← F_fs.y · 1/2 to img.y − F_fs.y · 1/2 inc F_fs.y do
5      for x_cur ← F_fs.x · 1/2 to img.x − F_fs.x · 1/2 inc F_fs.x do
6          curPos ← (y_cur, x_cur)
7          curWindow ← crop(img, curPos, F_fs)
8          if option =' max ' then
9              curExtr ← min(curWindow.intensity)
10         else
11             curExtr ← max(curWindow.intensity)
12         end
13
14         for y_filt ← y_cur − F_fs.y · 1/2 to y_cur + F_fs.y · 1/2 inc+1 do
15             for x_filt ← x_cur − F_fs.x · 1/2 to x_cur + F_fs.x · 1/2 inc+1 do
16                 F_f(y_filt, x_filt).intensity ← curExtr
17             end
18         end
19     end
20 end
```

3.1.2.3 Intensity stretching filter

With the help of the function, shown in equation 11, the bright intensities diverge from their dark opponents. Even for a human inspector it is hard to distinguish each *object* pixel and thus it is likewise problematic to define a clear boundary to separate both groups. The intention of this function then is to highlight the certain attributes. If the local intensity is higher than the average of the full image, then this difference will increase the current pixel intensity. Otherwise, if the difference is negative, for values beneath the average, the lower intensities are further decreased, due to the underlying subtraction. Since it is not constantly adjusting the individual elements with the same value, the image appearance changes and thus retrieves an information benefit.

$$F(x,y) = F(x,y) + \left[F(x,y) - \bar{F}\right] \tag{11}$$

with $\bar{F}$ as the average intensity level from the current filter position.

Algorithm 3: Feature extraction: intensity stretching

```
1  function intensityStretching (img, filtersize F_fs)
   Output: feature image F_f
2
3  F_f ← copy(img)
4  for y_cur ← F_fs.y · 1/2 to img.y − F_fs.y · 1/2 inc F_fs.y do
5      for x_cur ← F_fs.x · 1/2 to img.x − F_fs.x · 1/2 inc F_fs.x do
6
7          curPos ← (y_cur, x_cur)
8          x̄ ← getCurrentMean(img, curPos, F_fs)
9          for y_filt ← y_cur − F_fs.y · 1/2 to y_cur + F_fs.y · 1/2 inc +1 do
10             for x_filt ← x_cur − F_fs.x · 1/2 to x_cur + F_fs.x · 1/2 inc +1 do
11                 F_f(y_filt, x_filt).intensity ← (img(y_filt, x_filt).intensity − x̄)
12             end
13         end
14     end
15 end
```

3.1.2.4 Unprocessed image intensities

At first glance, using raw pixel intensities might appear as a weak feature, but its intention is a different one. When viewing an arbitrary ROI as a human, for most of the cases it is not too hard to distinguish at least a smooth border between *object* and *environment*. Unfortunately these impressions are not directly transferable to computer vision, by just reading the bare local image intensities. However, machine learning might have the possibility to learn the significant highlights. The worst case, that could occur is when a pixel intensity (more precisely a feature state) appears likewise in the *object*'s and *environment*'s area. But this could easily be classified as *unknown*, which simply states it cannot be safely classified with the raw intensity levels alone.

3.1.2.5 Correlation coefficient

The correlation is described in the beginning of chapter 3. While processing F_s, a matrix with the results is generated, where a value in all of the cells describe how likely the image center from F_r was found at the current position in F_s. Besides the position, it could also compare how similar the patterns are, in purpose of the cases where for instance a dark ROI differentiates itself from the brighter vicinity, which yields a different result as if when both regions are alike.

3.1.2.6 Combination of the stated features

To have a better impression, the options mentioned above could be stated as basic

modules. Whenever they are defined, a lot more new features can be generated by combining these modules. Subtraction, amongst other methods, is a lightweight operation where not too much additional time is lost on each pixel and a desired form of combination is achieved. For this certainly there are further options, where different approaches may retrieve other numeric values, but if these results still correlate with each other, then no useful information was added. Hence, a single combination-option is enough because it is more important to reduce the processing time. If two feature-images (i.e.: the filter response matrices) are subtracted cell by cell, various outcomes are expectable. For instance, it is aimed at to point out specific scene highlights (a similar effect as with the unprocessed image), or to mention a negative example: a noise like appearance.

3.1.3 Building the dataset

In the section above the feature extraction is shown. It is a basic part for constructing the dataset. But ML algorithms like the RF do not operate with these isolated values directly, they first need to be put into a constant order of a vector. Gathering the features into such a container indicates, this combination describes one particular state, which the algorithm uses either for training or testing.

Summarizing, a raw ultrasonic frame is retrieved initially, which in turn gets processed by any of the presented algorithms. The result is then again another image, where the intensity for each pixel is customized according to the selected function. At this point there is only a collection of processed images with equal sizes.

In order to retrieve the desired feature vector, one has to iterate through all feature-images and combine them with the appropriate label, which was artificially retrieved beforehand. To generate a such a vector ν for a given coordinate $P_f(y, x)$, a value is extracted from this specific position of a feature-image F_f and concatenated to ν. If this is iterated for all F_f, one ν is generated. This is summarized in equation 12 with N_f as the amount of feature-images

$$\nu(P_f) = \bigcup_{k=1}^{N_f} \{F_{f,k}(P_f)\} \tag{12}$$

This $\nu(P_f)$ can already be used in the RF prediction. Alternatively, if $\nu(P_f)$ is intended as part of the training dataset, the class identification is required to be attached, which is taken from the related element of the label matrix, described in chapter 3.1.1. To retrieve the full dataset, this procedure needs to be repeated for all coordinates and training images. This is visualized in table 5. If this information is created, the RF can already start with its training.

Table 5: The illustration of how the dataset is constructed with random values.

Position		Label	Feature 1	Feature 2	...
x	y	OBJECT	132	34	...
x+1	y	OBJECT	121	54	...
x+2	y	ENVIRONMENT	34	2	...
...	...	...	...	...	...
x	y+1	OBJECT	144	78	...
x+1	y+1	ENVIRONMENT	12	30	...
...	...	...	...	...	...

3.1.4 Pinpointing the coordinates of a ROI

When the user has selected the center of the desired tracking object, immediately afterwards the image segmentation begins to obtain the labels. The extracted patch around the ROI is then processed by N_f features, thereby multiple feature-vectors are generated, combined with the obtained labels retrieves the dataset.

Whenever the classifier finished its training phase, it begins to predict sample vectors already on the next frame. Since one is able to chose between an absolute classification and the numeric probability of each class, there are two different ways to go on from here. Either the classifier estimates the sample as *object* or *environment*, that could directly be transformed into a segmented image. Alternatively, the probabilities first are mapped on the possible intensity levels, which can be used to create a grayscale image. For applying a segmentation on this image, a threshold t_p needs to be set, that converts the results in to a deterministic prediction. In other words, the choice at this point defines, whether the classification is made by the RF alone, or if the probabilities are processed by suitable data separation tools further on (e.g. clustering, or even another feature extraction as described in chapter 3.1.2). A rather fair method for automatically defining t_p is the variant demonstrated by Otsu in [76], because instead of a plain average, the value is orientated at the histogram, that often gives a good insight of the available groups.

In any case, whenever the deterministic image prediction is available, the next step is to define the center of the *object*. If this is solved by calculating the arithmetic mean from y- and x-coordinates of all involved pixels, then there is the drawback of being sensitive to outliers. More secure is the declaration of a centroid (equation 13, with u,v as x, y coordinates respectively, Ψ as the set of pixels predicted as *object*)[77, p. 223].

$$\bar{y} = \frac{1}{|\Psi|} \sum_{u,v \in \Psi} v \qquad \bar{x} = \frac{1}{|\Psi|} \sum_{u,v \in \Psi} u \tag{13}$$

To further increase precision, outliers could carefully be removed. One information

about the tracking system is very useful for this special kind of filtering. Taking into account that an arbitrary ROI does move little between two consecutive frames, then a sample is assumed to be an outlier, if it is situated next to the image margin and is estimated as *object*, since in these areas ideally only *environment* should be detected. These false predictions can be neglected, when simply for instance the related label is switched. However, the more the outlier is located next to the center of the ROI, the harder it gets to decide whether it is a false prediction or not. Hence, a strict binary mask that insensitively predicts every pixel only as either valid or invalid is inappropriate. What fits the situation more is a gradient descent, such as a circular decrease of the influenceability. However a circle is not always a good representation of a ROI, when for instance some are elongated. It then presents as useful to make this descent relative to the shape of the object. Figure 18 shows an example, where the colored numbers represent the percentage of the actual *object*-prediction at the current position. For instance the RF-prediction was 255 (the brightest intensity on the image) but the referring reduction value is 25 % then the intensity is lowered to a darker gray of $\approx$ 64, as low that it in many cases won't be estimated as *object* anymore, which highly depends on the threshold function.

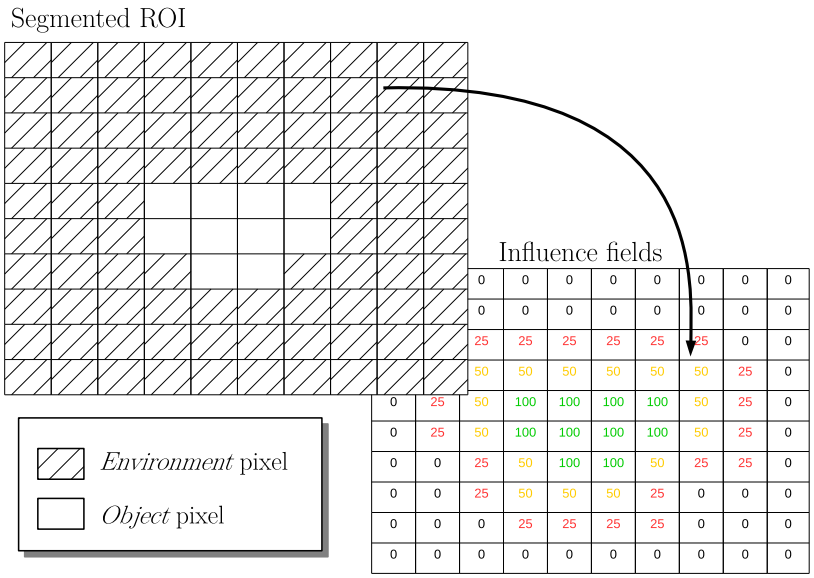


Figure 18: Example of a mask for an outlier suppression.

Such a mask quickly can be generated in a few steps, with algorithm 4. Basis of this function is the naïve segmentation as shown in figure 17 (chapter 3.1.1). Whenever on the binary image F_b, at position (y, x), an intensity $F_b(y, x) \neq 0$ was found and it's adjacent pixel e.g. $F_b(x+1, y) \neq 0$, as well as on the opposite side $F_b(x-1, y) = 0$ were

detected, then $F_b(x, y)$ gets the current percentage level, which in the end gets multiplied with the classifier's prediction result. Starting from the center, when moving towards the margins this percentage gets reduced appropriately step-by-step. The decrement of the percentage should be big enough, that the outliers at the margin are excluded completely (0 %).

Algorithm 4: Generation of the outlier suppression mask.

```
1  function generateInfluenceFields (segmentatedROI, percentageDec)
   Output: influenceMask
2  F_b ← copy(segmentatedROI)
3  for curPercentage ← 100 to 0 dec percentageDec do
4      for y ← 1 to y_max − 1inc +1 do
5          for x ← 1 to x_max − 1inc +1 do
6
7              if F_b(y, x) ≠ 0 then
8                  if (
                   (F_b(y, x + 1) = 0  ∧ F_b(y, x − 1) ≠ 0) ∨
                   (F_b(y, x − 1) = 0  ∧ F_b(y, x + 1) ≠ 0) ∨
                   (F_b(y + 1, x) = 0  ∧ F_b(y − 1, x) ≠ 0) ∨
                   (F_b(y − 1, x) = 0  ∧ F_b(y + 1, x) ≠ 0)
                   ) then
9                      F_b(y − 1, x) ← curPercentage
10             end
11         end
12     end
13 end
14 influenceMask ← F_b
```

An exemplary result of this procedure is shown in figure 19. The brighter intensities in the lower two images correlate with the probability to be part of the *object*. One remarkable note is the difference of the shown center in the sections, with and without the mask. This center is the relative localization of the ROI, which the algorithm will use to define as position for the current frame. Based on such an offset it's easy to see, that this outlier suppression can take a big influence on the prediction's accuracy.

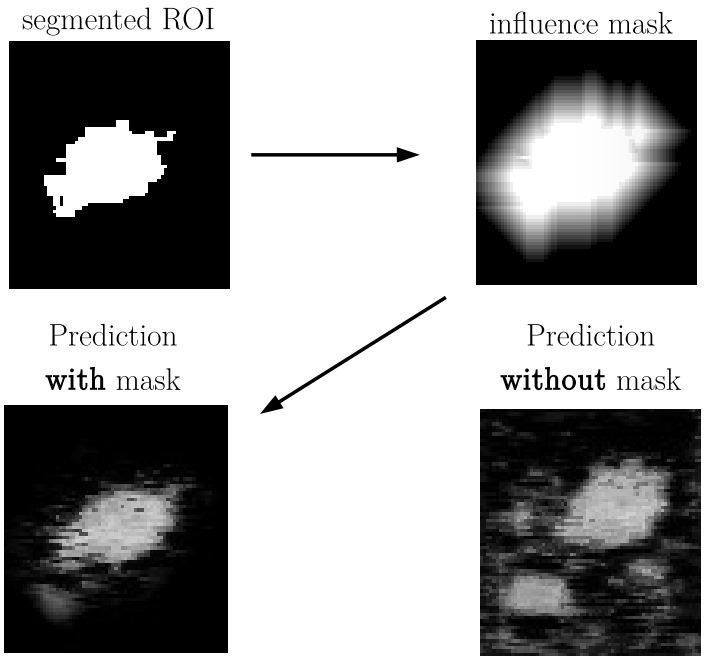


Figure 19: An example for generating a outlier suppression mask and its influence on the actual prediction result.

3.2 Landmark estimation

This solution approaches the problem in a different kind of manner. The algorithm is designed to analyze speckle-phenomena that occur during the process of an ultra sonic scanning. As mentioned in chapter 2.1.2, the appearance of those patterns are relative to their environment and thus a once learned trajectory should be stored, where these kind of patterns encode the position of a ROI on its trajectory. Certainly the involved parts start to move over time (e.g. drifting) and thereby the appearances vary. These unseen pattern refer to no known position. To increase generalization and being able to handle such data too, a RF is used for this encoding.

The chapter is structured into the implementation of:

- *rough-tracker* - that handles the speckle patterns.
- *fine-tracker* - a method suggestion to further process the result of the *rough-tracker*.
- *further optimization* - miscellaneous options to further optimize the precision.

3.2.1 Rough tracking

The main characteristic of the rough tracker is to follow the object in a steady, robust way. Due to a subsequent tracker that handles the inspection of the exact position, it is not important to retrieve a precise position in first place. But because of its further analysis, a rapid processing is required. This leaves more processing time for the complex fine tracking algorithms. In conclusion, the tracker:

- needs to provide a fast frame processing speed.
- needs to provide a robust position estimation of the object each frame.
- does not need to be accurate.

Unsupervised methods that rely on a relative argument as input variable (e.g the last position) have a risk to lose the object and get lost. Once they went astray it is *pure luck* for them to get back on track. This might happen when, under all possibilities it was chosen to stay at the exact same spot, which means when the object moves back within the reach of the tracker, it could continue its work. However empiric tests have shown it is not unlikely to drift away to the image's margin and remaining at this position, practically impossible to get back on track again. Which breaks the requirement of providing a robust position estimate.

Such an effect is avoidable, if the algorithm is able to forgo local ROI features and instead performing the tracking with attributes independent from the object. This way a current estimate doesn't get affected if there was a poor prediction of the preceding sample, that otherwise would spoil a correct estimate (e.g.: the tracker was moved out of reach, based on the previous false classification). Therefore, the RF should be trained out of the data from an initial short unsupervised tracking. Figure 20 gives an overview of the rough tracking algorithm.

3.2.1.1 The initial run

First of all the trajectory needs to get estimated. By the algorithm's ignition, a localization of the desired traceable ROI is demanded. Along the CLUST competition this is given for the first frame by the annotation coordinates for a specific sequence and object, where in a real application the user simply needs to set the initial ROI coordinates. With this information the object's reference patch F_r is extracted once from the first frame, which is the ROI itself for the fine tracker. Simultaneously, from the same image, the training patterns (i.e.: speckles) for the first position of the object's trajectory are recorded. These pattern localizations can arbitrarily be defined, but need to be at a fixed position during the tracking period. The only limitation on selecting these extraction areas, is to ensure that the underlying pixels carry the ultrasonic data.

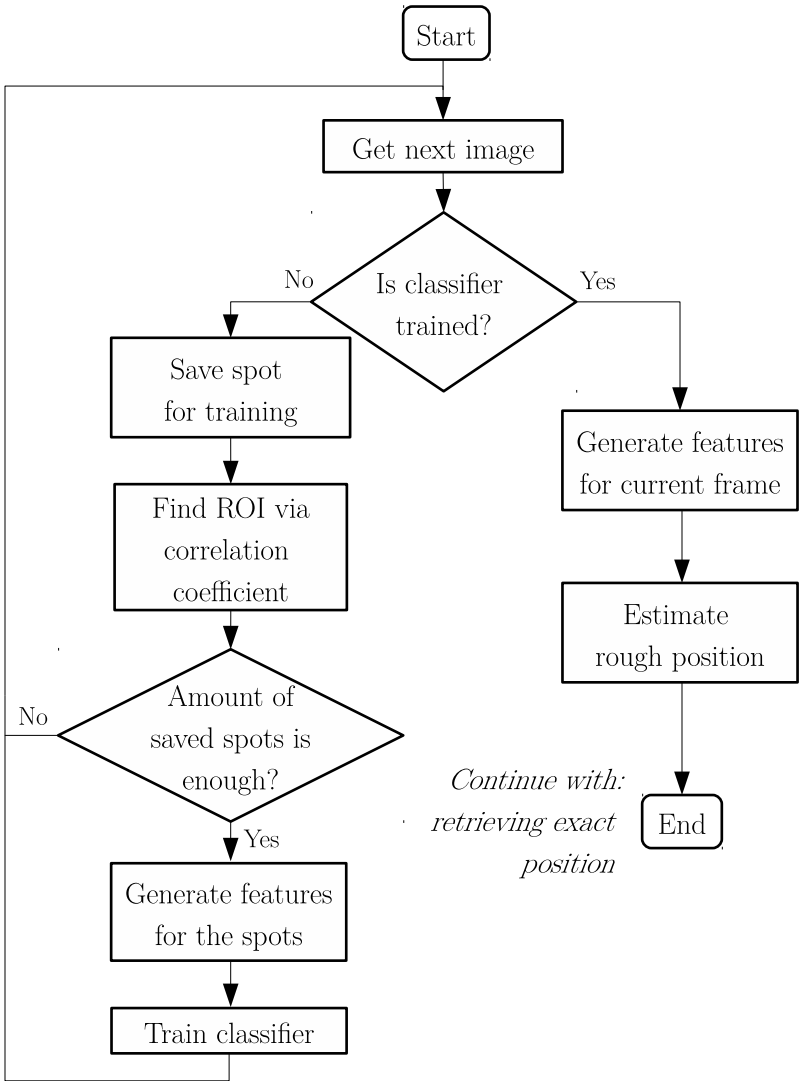


Figure 20: Overview of the rough tracking algorithm, passing its estimated position to the fine tracker as an input parameter.

Based on the initial coordinates and the saved F_r, the ROI is searched for in the current image-neighborhood but this time in the upcoming frame. For this the *correlation coefficient* is used (as explained in the beginning of chapter 3), with a certain constant search window size F_s. Once this is achieved, the patterns are saved again for the same purpose as before. In other words, in each iteration (i.e.: frame) of the initial run, on the one hand patterns from a constant location are extracted, where on the other hand the correlation retrieves the object's new position. This procedure is repeated until this unsupervised tracking has collected enough of these frames for the image-vector I_{init}. It stops whenever the ROI has collected at least $n_{init,min}$ frames and was passing by its initial position 4 times, which usually means that 2 breathing cycles have happened. For this purpose, on each frame simply the correlation result from the first ROI estimation is compared with the one between F_r and a patch of the same size, extracted from the tracking object's initial position. If the current patch results a higher score than the initial one with a small tolerance it indicates, that the ROI just moved over its starting position. Then this check is locked until the euclidean distance between the current ROI position and the initial coordinates is higher than 5 pixels again, to prevent double checking in a short time. In any case, the unsupervised tracking finishes latest after $n_{init,max}$ iterations. This way the trajectory is recorded step-by-step and the classifier is ready for its training in the next step. The procedure is summarized in the figure 21.

A few points have to be considered for the integration of this mechanism. There are generally different speed options for a traceable object, which encounters the algorithm not directly as a time problem, but as smaller or bigger steps of motion. This behavior can be controlled by the size of the correlation-search window F_s. If the shift is minimal then the dimensions of F_s are also allowed to become appropriately small, then again a quicker ROI motion naturally requires a wider dimension. Tiny search areas enable a faster processing speed and arise probably less false predictions, due to the lack of the higher amount of potential results in general. Despite the timing factor, it is more stabilizing to have a wider scanning range along slow motions, thereby the system is better prepared to handle situations like occurring *jumps* (i.e. there's a higher ROI position difference from one frame to the next), when for instance a real-time environment of the image acquisition isn't guaranteed.

3.2.1.2 Analyzing the Speckles

Due to the reproducible attitude of these patterns, it would be easy to hold a sample per each ROI-position and simply figure out its position by comparing this database with an arbitrary pattern. Often simple approaches consume less processing time, which is needed for a real-time application. Such a simple method for comparing data containers is for instance the *Hamming distance* (chapter 2.2.3: other performance measurements). In this case it would inspect the patterns of the training dataset to compare each of them with the current sample. Due to its algorithmic nature one is able to boost the

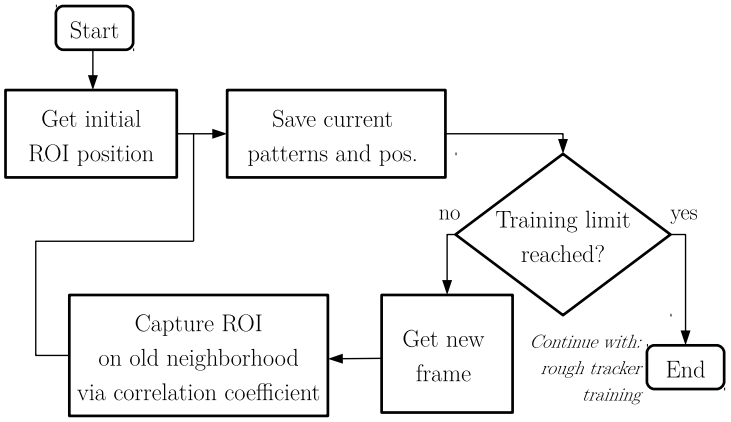


Figure 21: The course of the initial run, where the data for the rough-tracker is gathered.

processing speed by parallelizing the pixel-comparison either on a GPU or by multiple CPU cores. Utilizing this metric appears fair enough, however there are a few reasons to avoid this technique in the given scenario. A bigger database with a lot of samples might increase the accuracy, but it also increases the effort to inspect every entity. But even worse the appearance of a given sample varies over time and thus the recall value with the patterns of the dataset decreases. As consequence, a diminishing score of the Hamming distance is expectable, which in the end lowers the actual prediction robustness.

Integrating a RF eases both of the timing and robustness issues. The classifier is trained on the whole database of patterns, so it is able to figure out the significant parts of a pattern to improve the recognition of unseen data. Internally, its decision trees can be designed in a way, for as a result the ROI position is returned.

For this a dataset is needed, where the label for one feature-vector represents a position on the trajectory. That vector itself is solely composed of the related intensities in a manner, that for each of the pattern's pixel a separate feature is added. It is critical, that an extraction of a given feature in the training and testing phase is always at the same position (i.e. a feature does not only provide a pixel intensity but also carries a local information). To provide a more generic algorithm that runs not only on selected data but also when changing arbitrary influence factors (e.g. the ultrasonic scanner), it comes in quite handy to select multiple patterns from different areas. This avoids algorithm confusion, when in an observed area high-intensity differences occur but from a more direct source (e.g. movement of a bone), instead of the expected background granulation. An example is given in figure 22. The brown boxes define the areas where the patterns are extracted, as the colorized squares represent a pixel/feature of this method.

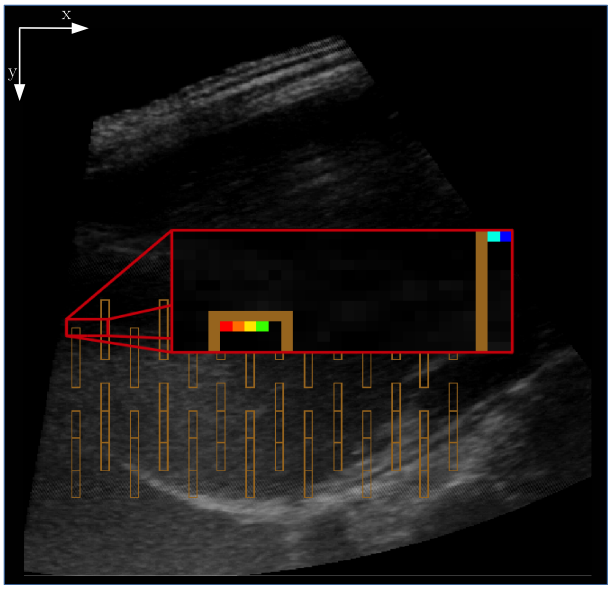


Figure 22: The feature extraction for the rough tracker.

To give a better illustration of the process, an example dataset with artificial entries is shown in table 6, where every feature represents an intensity from a specific location (i.e. a certain pixel) and each marked with an individual color (Feat. = Feature). The feature values in this example were taken from the ultrasonic frame shown in figure 22, where the marked pixels stand for the feature extraction at the correct position with the same color in table 6. More precisely, collecting these entities is generating a vector, when on one image the red pixel is stored as Feat. 1, the orange as Feat. 2, and so on. Each vector then represents the pattern states for a certain frame, thus the position label, which is provided for every frame by the unsupervised tracker. Certainly assembling these containers retrieves the dataset for the training.

Table 6: The representation of the speckle-pixel in a dataset, to train the rough-tracker.

ROI-coordinates	Intensities of the ultrasonic pattern appearances							
Label	Feat. 1	Feat. 2	Feat. 3	Feat. 4	...	Feat. n	Feat. n+1	...
[x , y]	68	82	86	81	...	57	56	...
[x+1, y]	70	80	84	81	...	54	56	...
...	...	...	...	...	...	...	...	...

A more detailed description of the rough tracker's dataset composition is described in appendix A.2.

3.2.2 Fine tracking

The essential characteristics of this algorithm part is to retrieve the exact object localization within a constricted area, considerably smaller than the size of the full ultra sonic image frame. From here, plenty of ways can be chosen to go on from here, such as learning the ROI pattern with the help of ML-algorithms, that compare the representative patch F_r with the current frame. Or using more complex approaches with feature detectors, descriptors and matchers like SIFT [78], which was already evaluated for the CLUST challenge in [10]. In ML, CNN on the one hand have quiet a good reputation when it comes to recognition, on the other hand they ordinarily need more time in training. In [9] it is mentioned that their approach required 1.5 h for adjusting to the CLUST data. Yet a RF is trained incomparably faster and is known to handle unseen data well. When it comes to a direct matrix comparison, a rather robust method is the *correlation coefficient*.

All of these methods have one attribute in common. It doesn't matter how ingeniously their design is, non of them performs without mistakes. This being said, it has to be inspected how deep an impact of a false prediction at this position could get. If the FP and FN of a very reliable method is at a dwindling low level, it could be considered that these cases are just going to be ignored (means their false predictions are blindly accepted in any case), since they seemingly barley ever occur. In contrary, if the algorithm is not that reliable or any false prediction is not acceptable at all, then another safety check needs to be included. Leaving these decision up for a human inspection may greatly decrease the FP- and FN-rates, but certainly is impractical to apply. Thus a control switch close to this state is desired, that doesn't affect the working flow much.

Recovering hypothesis analysis from chapter 2.2.4.2, then significance testing seems to be an appropriate tool, that on one hand decreases the false predictions as on the other hand is kept simple and lightweight and thus will not increase the processing load too much. As it is stated that a result of the fine tracker may or may not be correct, there are yet two cases to consider. However, deciding whether the estimate was right or not does still not show an appropriate default fallback to chose in such occurrences. For this there a two options. The first, is specifically concerned about the mechanisms of this two-component algorithm. Since the rough tracker is designed to deliver a preferably robust but not exact position, it is appropriate to chose the rough-trackers position as default location. The other option depends on the assumption that in the usual case no big movement can be expected, between two consecutive frames and thus simply the last position is selected. Unfortunately this comes with the drawback, that this doesn't cover the few exceptions of a delay caused by the system, where wider jumps in movement are potentially detectable. Significantly, the first option stands for a more robust yet less accurate solution, since the rough tracker is capable of such jumps but half of a rough-tracker estimate range has to be expected as usual error each time. In turn the later option is a lot more precise, due to the small change of the distance between two consecutive frames for most of the cases, but might retrieve an error a lot higher during

a system delay. Nevertheless the probability of such interrupts should be observed as a rarity, otherwise the user, perhaps, wouldn't be able to utilize the ultrasonic scanner at all. Additionally, considering a case where a jump occurred doesn't necessarily mean the fine tracker is not able to estimate the correct position. In other words, that the two probabilities (a jump occurred and the tracker's capabilities were exceeded) summing up is an uncommon event, unlikely to happen that often. Hence, the precise solution is selected for the algorithm.

In conclusion for retrieving a current frame position, there is the default hypothesis H_0, where the algorithm should use the last sincere prediction from a previous frame. As against H_1 is the most recent result of the two-component tracker from the current frame. Thus H_{cur} defines the selection, which is used to continue the algorithm. What is still missing for utilizing the significance test is the rule, when H_0 is going to be rejected over H_1. Since the rough tracker's RF does retrieve a class probability score RT_{Score}, the algorithm is able to verify the outcome of an arbitrary sample prediction with the help of this value.

After the algorithm finished the initial unsupervised tracking, the first ROI-position is estimated by the rough tracker immediately. Along to this prediction the desired score is created, because the ultrasonic pattern appearances from the current frame marginally are different, compared to the ones from the rough tracker's dataset. The related class prediction probability then is held as some highest achievable value. The more an arbitrary pattern appearance differs from the ones learned out of the generated training dataset, the lower this value becomes. Therefore a limit l_{thresh} needs to be defined relative to this optimal value. The helper function of this control switch is described in equation 14.

$$H_{cur} = \begin{cases} H_1 & \text{if } RT_{Score} > l_{thresh} \\ H_0 & \text{otherwise} \end{cases} \tag{14}$$

As stated in the beginning, the fine tracker offers many possible implementations. In the following the implementations of a ML solution, the Random Forest-recognition, as well as a matrix comparison with the correlation-coefficient comparison, are demonstrated.

3.2.2.1 Comparison with the correlation-coefficient

The basics of this technique is already explained in the chapter 3. The more one chooses to rely on the preceding rough-tracker, the smaller the sliding-window-dimensions of the correlation possibly can get, as a result the processing speed increases. Considering a case, where the rough tracker's prediction never fails (i.e. the precise location of the ROI is always within the range of the current predicted label/position), it is then appropriate to set the window's dimension similar to the estimated class range. But because this usually counts as an ideal state, it might be more appropriate to gently exceed these limitations, in order to capture the possible false predictions.

3.2.2.2 Random Forest recognition

Other to the method mentioned above, this option doesn't need to handle every pixel of the reference patch. It extracts significant parts of the patch, thus it's aiming to handle the forthcoming unseen ROI well and because of the lesser work per frame have a faster processing time. More precisely this approach is similar to the technique shown in sub-chapter 3.2.1.2.

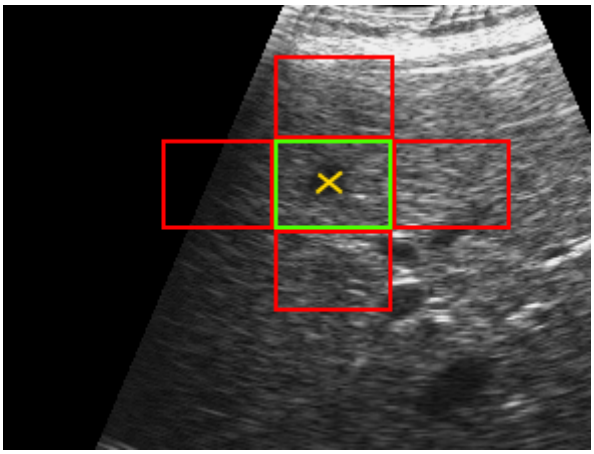


Figure 23: The regions dataset extraction for the RF-recognition.

The reference patch F_r is extracted like before and each of its pixels are used as a specific feature (here again one feature carries the intensity level as well as a localization information, then again it is crucial that the features for each iteration are extracted always at the same relative position as given by the training data). The features for the *object*-label is extracted directly from the ROI itself and also in its nearest pixel-neighborhood. Feature vectors for the *environment*-labels are generated almost artificially, as an arbitrary region is chosen for this. It is recommended to set this choice close to the *object*, to make it clear for the classifier to not only learn the significant part of the tracking object, but to clearly distinguish between these adjacent patterns belonging to the *environment*. This procedure is visualized in figure 24 for the whole trajectory and in figure 23 on a real ultrasonic frame, with the green area as *object* and red as four *environment* areas, where the ROI is denoted by the yellow cross.

Once these patches are captured, the classifier is trained right away, since it will already be used for prediction in the second and all following subsequent frames. The prediction itself is also applied like the method before with a sliding window, the difference here is that for each cell of the result matrix a class probability is given from the RF, about the ROI-center, instead of the correlation coefficient.

After the position in the second frame is located the procedure from the beginning is repeated, gathering more training data and retrain the forest for its next prediction. This loop is continued, until the rough-tracker is ready.

As already mentioned, this approach is intended for the classifier to especially focus on the significant parts, that ensures its generalization. With this demand it also should be feasible handling uncategorized pixels states. For instance image data originating from a different source than the ultrasonic scanner. This could be the black area surrounding the actual ultrasonic scatters, or other data artificially added by the image provider (e.g.: text or a diagram with the hint which part of the body is observed in the record). When for instance the ROI is near to the edge of the scanner's range as shown in figure 24, then during the first iteration, the left *environment* area extracts a part of the non-ultrasonic image data, which is used for the classifier's training in the end. However even if non of these kind of training data is available, the RF still should handle the situation reliably, as it has sufficient data that never included a black area as part of the ROI.

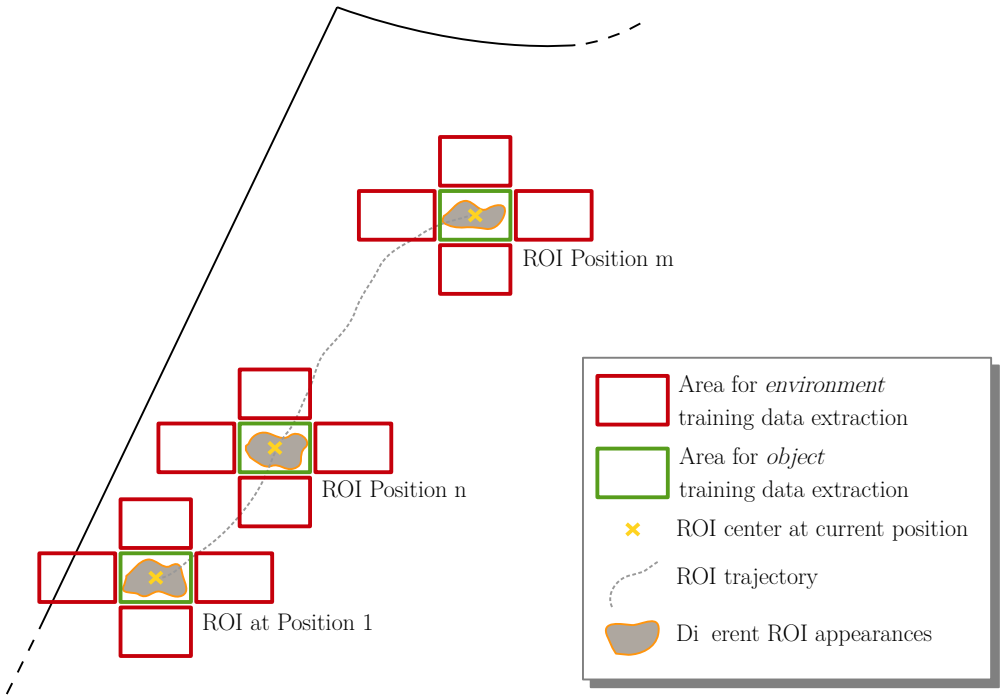


Figure 24: The procedure of the RF-recognition.

3.2.3 Further optimization

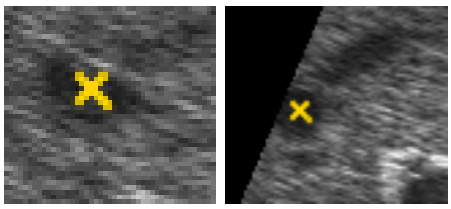
With the demonstrated methods there are still a few special cases open, able to worsen the accuracy. For this, the following section demonstrates further options, designed to capture possible issues, which the base algorithm lacks to handle. Common problems are:

- The classifier has learned an insufficient trajectory (i.e. the trajectory in training is shorter than in the subsequent testing phase).
- The trajectory drifts over time, thus leaving the learned path.
- Black non-ultrasonic image data occurs at the margin, that possibly confuses the algorithm.

Chapter 4 elaborates their efficiency, thus subsequently only the principles of these techniques are explained.

3.2.3.1 Black area detection

Whenever there is non ultrasonic image data, it possibly confuses the presented algorithms. The impact for the *correlation-coefficient* method is arguably higher than for the RF variant. While each affected pixel, unfortunately, has an equal weight on its result with the first method, only a little selection of image points are required for the second one, however certainly will fail if they were taken from the area which is going to be occluded.



(a) Frame 1 (b) Frame 188

Figure 25: A given ROI from the MED-1-1 sequence, at two different locations from its trajectory.

A basic scenario is shown in figure 25, where the traceable ROI is marked with a cross. The correlation method is comparing each region during testing with a pattern, taken from the tracking ignition (F_r), shown in figure 25 a). The more the object-localization reaches the black margin, the more pixels of the current image with $F_c(x, y) = 0$ are

compared with corresponding $F_r(x, y) \neq 0$ from the reference pattern. Normally, intensities are examined that have a much higher level in both cases (correct and wrong area). Due to the black levels, the recognition is constantly worsened even though these pixels should not take any influence on the decision. This situation eases other surrounding areas to be identified as object, over the actual ROI (such as regions with less non-ultrasonic image pixels), when even though the score there would be low in general. One reason is, if the difference between the scores of the false ultrasonic image pixels is smaller than compared to right but occluded ones. Tests have shown, that in this particular case the algorithm got confused as its prediction shifted to the surrounding ROI-like area, ending in a false learning of the object's trajectory. Whereas the other suggested ML-option, either also fails because its prediction relies on the occluded pixels or is luckily capable of handling the situation, when for this case the decisive image data is extracted from the bottom right (i.e. it doesn't even get in touch with the problematic zone). Conclusively, for related scenarios this behavior takes a big impact, that quite easily could even prevent a successful tracking in general.

For avoiding these situations, another routine checks black pixels (i.e. $F(x, y) = 0$) and interpolates their intensity to a more plausible level. Since the goal is to retrieve a possibly high score for comparing the current F_c with the initial F_r, the interpolation is leaned on the initial appearance. The corresponding pixels are copied from the reference pattern to the current ones with the black area. This way, all the affected coordinates receive a full score, with the intention that a ROI around the non-ultrasonic area is more likely to be chosen than its adjacent regions, which look less alike but contain more ultrasonic image data over the whole pattern. The process is demonstrated in figure 26.

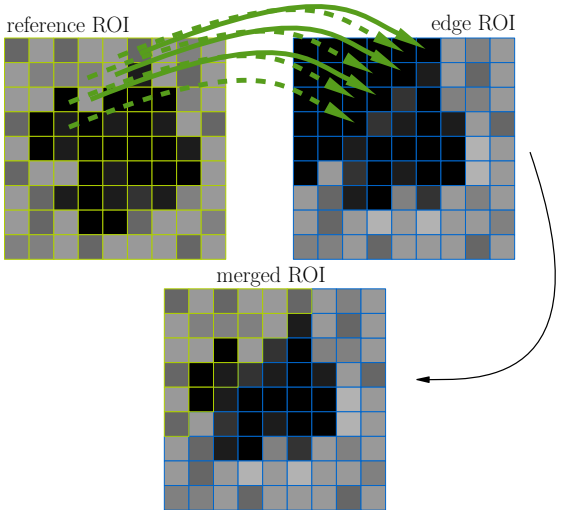


Figure 26: The black area is filled with the intensity levels from the initial ROI pattern.

3.2.3.2 Multiple rough classifiers

When working with different ultrasonic devices, the image structure is likely to change as well. Exemplary occurrences are differences in the look of the speckle phenomena, a darker or brighter image and a smaller or wider area for ultrasonic image data. During observing the behavior of the rough tracker, empiric tests have shown that its prediction quality depends on the pattern extraction areas, which in turn is influenced by the general image appearance.

Either the user has to manually find out the most fitting region before the actual usage, or more comfortable training multiple forests at different positions. Hence, during the testing phase, the algorithm uses with this *poly random forest design*, under all predictions from each of these classifiers the one, who holds the highest probability score for its result.

3.2.3.3 Shift trajectory

Sequence MED-1-1, showed it might occur for the object, it is not always moving along the same path. Here the object always followed the same trajectory but adds an offset in a specific direction over time, thus drifting away. The consequence for the rough tracker is simply that its prediction might be correct at the very first frames, but because of the constant offset is far away from the actual ROI-center, so the distance between estimated and annotated position (i.e. the error) increases. Thus within a loop, after a certain amount of frames $|F_{shift}|$, the difference of coordinates from the rough and fine tracker are calculated and the arithmetic mean of these samples generated. This mean value is taken as the current offset and simply added to the coordinates for every trajectory landmark (i.e.: a trajectory coordinate behind a feature vector label).

3.2.3.4 Artificially extending the training database

The main task of the rough tracker's initial run is collecting for each frame the position estimation and the current patterns. In this scenario, the worst case that could occur for creating a dataset is when there is only one feature-vector per position available, which usually results in a bad classifier's accuracy. However, there may or may not be a few spots with a bigger dataset if the same position was captured more than once, which solely happens when the object is passing by again on its trajectory back where it just came from. Even in the best scenarios the dataset still remains at a sparse state.

As mentioned in the beginning, the rough tracker does not need to work very precisely for its localizations. It is then tolerable to neglect the correct position estimates, as collected in the training phase. However, the estimation security is a lot more important, that enforces the right and disregards false *label*-predictions as best as possible. Hence, more feature-vectors for a given position can be constructed if the adjacent positions

are merged to a new artificial position (i.e. a new label) and their features kept as they are. This means now there is only a vague trajectory stored, as the possible predictable labels have decreased. Each of these equidistant spots include all possible feature vectors within their specific range. The mentioned range is relative to the trajectory's length and is simply the half distance from one merged trajectory landmark to the next one, in a manner that no data is used twice, which improves the dataset quality when two identical feature vectors with different labels can be avoided. This is visualized in figure 27, with the red line as a vague trajectory, gray crosses as the stored positions from the very initial run, black crosses as the new merged landmarks, where each feature vector (underlying to each gray cross) is now referring to within its specific range, which is depicted as the orange circles.

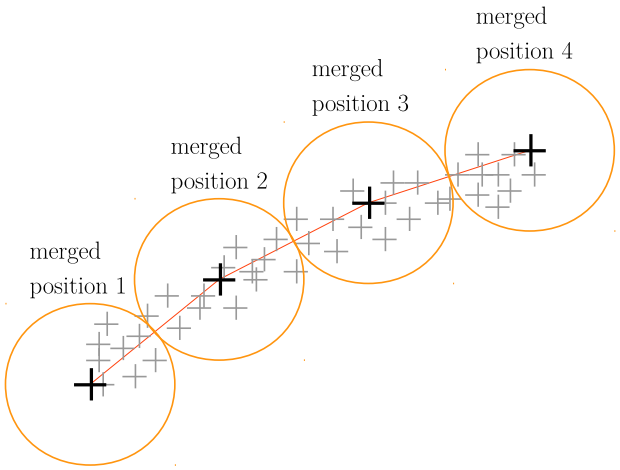


Figure 27: A merged trajectory compared to its originally recognized spots.

3.2.3.5 Adaptive trajectory learning

A less coarse variant is stepwise extending the decision database, by never ceasing the learning process even during the testing phase. Whenever the algorithm has learned enough of new input, which is after $n_{samples}$ newly added feature vectors, the rough tracker's random forest training reignites (i.e.: adjusting the decisions and predictions of each tree). After the rough tracker finished its adjustments, it is capable of predicting the most recent state of the trajectory's way-points.

More precisely the algorithm's training stage doesn't get adjusted, whereas its testing phase only gets extended. Whenever the classificator is ready to predict, figuratively spoken it is placing landmarks along the trajectory with a certain distance to each spot (as shown in figure 27). However the classifier is dealing with probabilities and thus false predictions are expectable. This estimate then needs to be assessed if it was valid

or not, so that the new data could be stored. As there are many labels to guess, a good (or an assumed correct) position statement then is whenever the fine tracker was able to retrieve the ROI within a specific range r_{lm}, with its center upon the currently estimated merged position, which is the rough tracker's prediction. This way the error cannot be excluded because it also cannot be guaranteed that the fine tracker is working flawlessly, but negative influence is minimal since the accurate position is not needed, as the coarse is sufficient enough for updating the landmarks. As a last step, whenever the new dataset is big enough, the rough tracker training can be reignited.

For the RF is able to recognize and estimate the extending trajectory, further spots are needed at least on one of the trajectory's ends. When it then comes to retrain the classifier, the two spots with the highest distance from each other are detected (S_1, S_2). Then the first new merged landmark is set upon one of these spots. Since there is always the same amount of merged landmarks n_{lm} for the rough tracker, their influence range is simply recalculated in two steps. First n_{lm} equidistant merged landmarks are placed from one end to the other starting from S_1 or S_2. Afterwards the real estimated spots are assigned to the landmark to which the euclidean distance is the shortest. Finally the coordinates of the merged landmarks are recalculated according to the mean value of their assigned spot-coordinates, which helps to rebuild the trajectory instead of creating a straight line. Empiric tests have shown that, whenever there is a case with a growing trajectory, the exceedance usually happens rather slowly, in other words, it takes multiples of such cycles to exceed a merged spot. Thus after a few turns, one is able to update the trajectory and can now successfully capture new positions with the rough tracker in future.

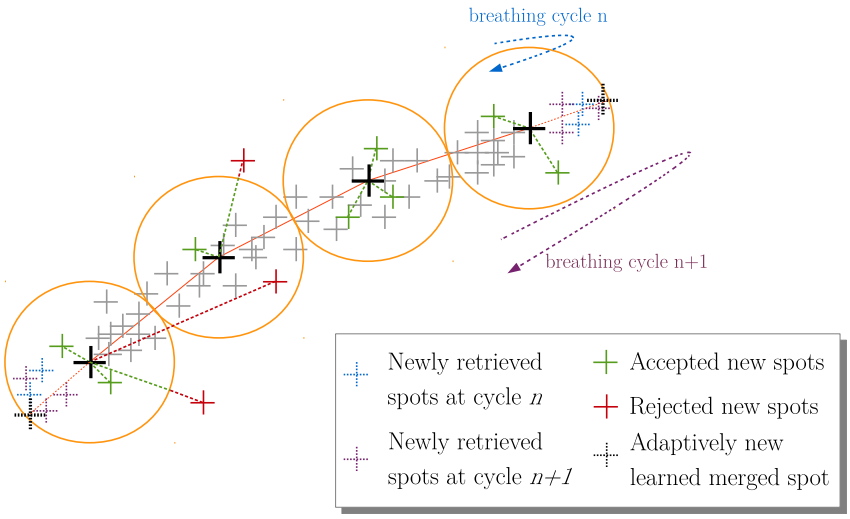


Figure 28: Visualization for the possible states of the adaptive range.

4

Evaluation

After all relevant methods are described in chapter 3 before, in this successive section the impact of the parameters on the actual tracking error and the strategy to define the optimal parameters are discussed.

4.1 Fundamentals of performing the algorithm's evaluation

In the context of ML, the presented algorithm and its settings are also known by the terms *model* and *hyper-parameters* respectively. Such models usually consist of many variables that take influence of the performance, likewise in a positive and negative kind of manner. With the help of a so called *hyper-parameter optimization* (or *parameter tuning*) it is pursued to determine the configuration, where the related model retrieves the best results (e.g. the highest accuracy).

The most commonly used variant of this optimization is the *grid search*, that atomically evaluates every possible combination, where for each added parameter the amount of joint values grow exponentially. This means, with many variables this method could easily become too time consuming for real applications. In contrary to this meticulous approach, a randomized search from a uniform density of the same configuration space retrieves a considerably well model after only a few evaluations [79]. In turn, a method based on probability theory is the *Bayesian optimization*, that approaches the minimum error based on the history of previous evaluated configurations and chooses the most promising hyper-parameters for each iteration [80]. Because the random and Bayesian approach seemed a bit too risky for a limited evaluation period and grid search could consume too much time, a combination of the random variant and the grid search were applied. By stepwise defining the parameters (e.g. the rough tracker separately from the subsequent methods), the amount of necessary test can greatly be decreased. Candidates for parameters like the position of the speckle extraction areas are randomly estimated.

In case of the tracking scenario, a model configuration is considered good, if the distance between estimated and annotated ROI-position is minimal. The related metric is the euclidean distance d_e, as described in equation 4a in chapter 2.2.3. Since the tracking is performed on planar ultrasonic images, the actual distance for the evaluation is based on two dimensions (i.e.: the y- & x-coordinates). The final formula for the error determination is shown in equatoin 15.

$$d_e\Big(P(y,x),\hat{P}(y,x))\Big) = \sqrt{(P_x - \hat{P}_x)^2 + (P_y - \hat{P}_y)^2} \tag{15}$$

On a closer look it is equivalent to the Pythagoras theorem, where $\hat{P}(x,y)$ is the algorithm's estimated and $P(x,y)$ the annotated position.

4.2 The ultrasonic data

For tracking on liver ultrasound images, MICCAI Clust has established a challenge for a direct comparison of different methods. This challenge provides 64 sequences for the

2D-tracking, who were recorded by 5 different ultrasonic scanners, where the length of a sequence range between 4 s and 10 min. The data is separated in a training- and testset, where each challenge-participant receives the trainingsset's annotation for the own development. This annotation determines the ROI position on many frames (but not all), which were manually defined by 3 human observers. The accuracy for the testset is evaluated by the challenge organizers themselves [6].

In the area of ML there are strategies, that help to improve the algorithm's performance. Especially when it comes to classification tasks, a commonly used technique is the cross validation (see chapter 2.2.4). Because both algorithms rely on classifying either pixel labels or trajectory landmarks, it is then appropriate to integrate this tool into the development. It was intended from the very beginning to set the algorithm's design with these techniques. Unfortunately, this is barely feasible due to the handling of the data during tracking. When working with a deterministic dataset (e.g. handwriting), usually it is randomly split into a part for training and one for testing, afterwards it is evaluated how well the classifier performed. For both of the own algorithms this is out of reach, because the algorithm is trained and adjusted every time an arbitrary tracking task is initiated (i.e.: there is no separated training part at all). Additionally for these algorithms, a given estimate always depends on the first image of a sequence and also on the previous tracking result (e.g. when trying to find the ROI around the previous ROI-position estimate). The situation is different in case of handwriting classification, since there the individual samples are usually is provided (e.g. MNIST) and doesn't need to be constructed beforehand.

4.3 Pixel classification

Classification of pixels that in the end retrieve a binary image, where the *object* is separated from the *environment* seems to be a promising approach, when considering the prediction quality that could be extracted from the related pixel-precise accuracy. Additionally, this variant even provides multiple options to determine the ROI position (e.g. the center or centroid of predicted *object* pixels).

The evaluation showed many different results that give an insight about the algorithm's attributes. For instance the tracker was able to obtain the ROI's position, with a comparably well tracking error already with its set of basic features. Figure 29 shows the result of sequence CIL-1_2, with the top graph as d_e for a current frame and below the three different kind of error measures.

To give an impression about the actual classification, a screenshot, taken out of a running tracking procedure, from two different sequences are shown in figure 30 (left: CIL-01_2, right: MED-02-1_1). With the top left accumulation of white intensities, as the pixel classified as *object*, which is the same as the purple group who are situated upon the estimated ROI position. The yellow cross denotes the center retrieved by the algorithm, where the blue cross is the annotated center for the current frame. Both images were

provided from two different ultrasonic scanners. Similar results of different scenarios are shown in appendix A.1.

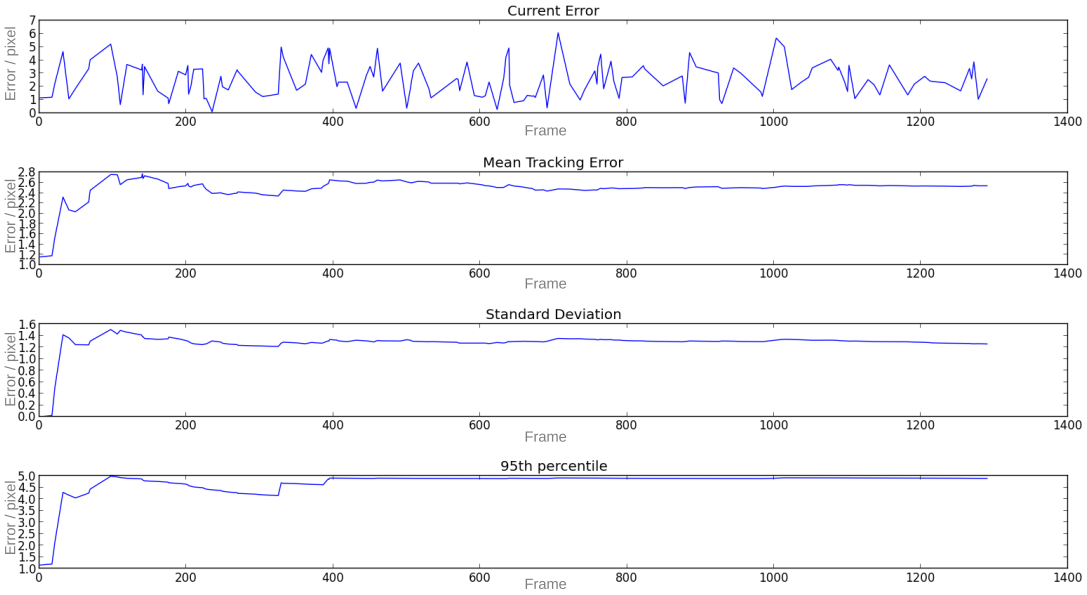
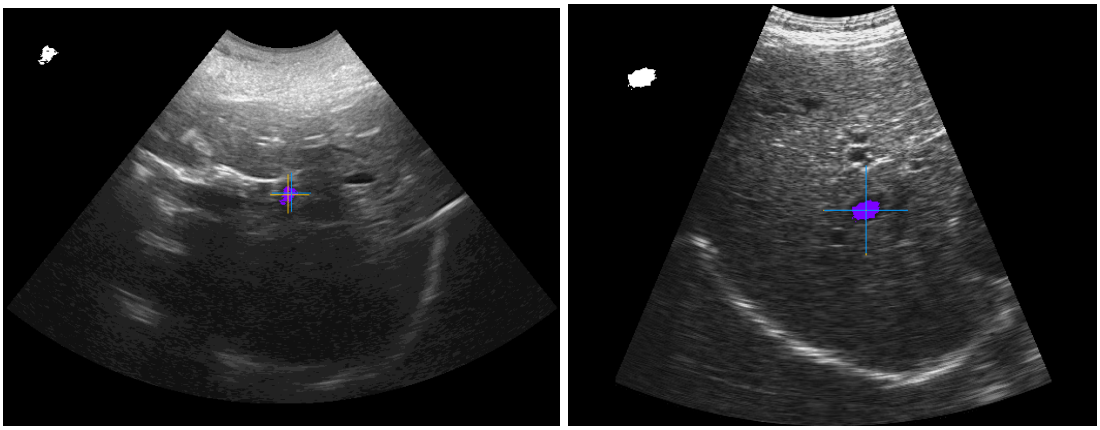


Figure 29: Tracking result of CIL-1_2 via pixel classification.



(a) CIL-01_2, Frame 142

(b) MED-02-1_1, Frame 2429

Figure 30: Exemplary ROI positions, estimated by the pixel classification method.

However in many cases the algorithm simply has lost the object during tracking. Figure 31 shows the results where just before the 1000^{th} frame the tracker moved towards the image margin.

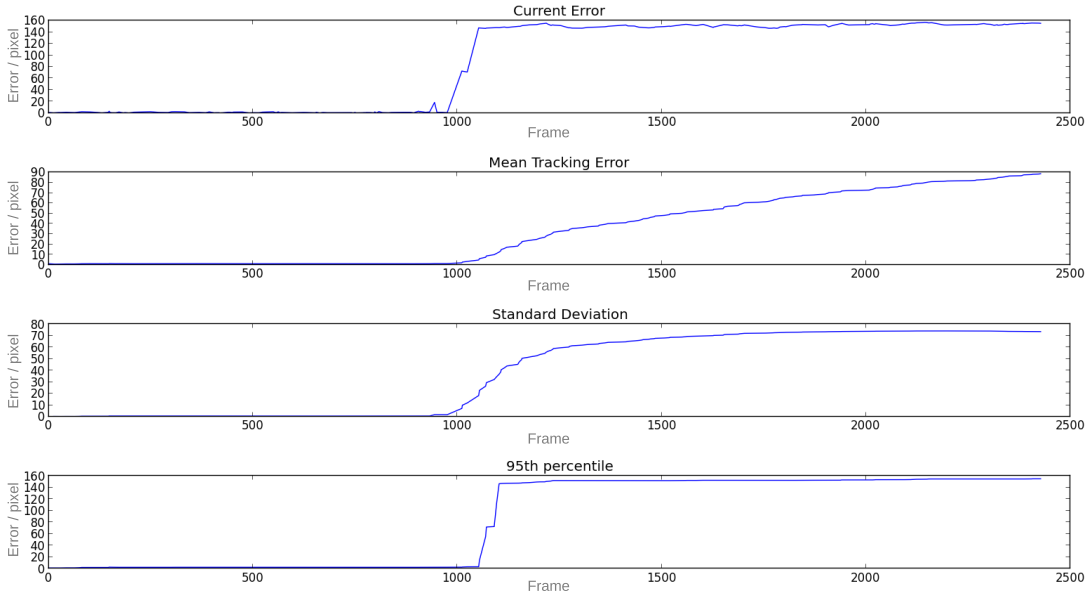


Figure 31: Tracking result of MED-02-1_2.

Another example is shown in figure 33, where the tracker mistakenly classified the adjacent region as ROI, hence the minimal constant offset that starts right before the 500^{th} frame. But in the end the algorithm couldn't be saved from going astray. For a better insight the object is shown in figure 32.

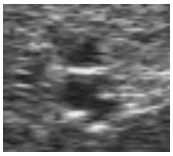


Figure 32: The ROI of MED-02-1_3, where the adjacent region has a similar appearance.

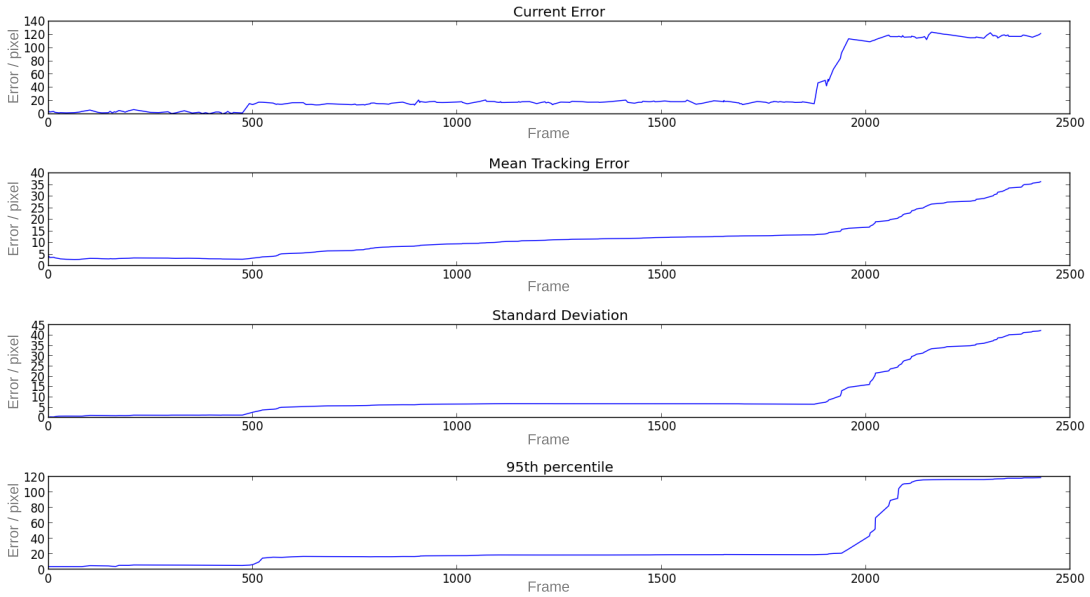


Figure 33: Tracking result of MED-02-1_3.

In conclusion, tracking via pixel classification is feasible for many cases, but has its problems that need to be eliminated for a secure tracking. Because of the adjacent pixels were falsely classified, the tracker has lost the object. Simply, because of their overwhelming majority, the pixel mass center point shifts away from the ROI over time. This is because the involved features apparently were not significant enough. False classifications cannot be avoided and likewise it cannot be prevented for the tracker to go astray. This clearly argues against the robustness and thus is less suitable for applications in the medical area, which discards the approach for further development. However, it was intended to design the model with the help of a ROC curve (see chapter 2.2.3) between the classifier's pixel estimation and the initially generated binary label image, as described in chapter 3.1.1.

4.4 Estimating landmarks

This approach solves the main problem of the pixel classification. Whenever the classifier is trained, the tracker is automatically orientated towards the object's trajectory and thus it is able after each poor classification to obtain an optimal result already in the subsequent iteration again.

Testing the algorithm was performed in parallel on multiple computers with the same system specification. Unfortunately, the same PCs could not be kept until the end of the hyperparamter-optimization. The results are reproducible if a test was repeated with the same computer. When switching to the new group of PCs, the results compared the

other group differed. Calculating the standard deviation of the difference from 84 tests, between a pc of the new and one from the old group, retrieves ≈ 0.7979 pixel for the mean tracking error, ≈ 0.5486 pixel for the standard deviation error and ≈ 1.9853 pixel for the 95^{th} percentile (see table 14 in appendix A.4 for the results of this comparison), in other words, the variances are minimal and thus the results do correlate. The processing time for evaluating the training set ranges from less than a minute until a few days. Because of the lack of time, it was not feasible to reexecute the gridsearch again with the new computer group. However, because their individual results are similar to each other, both grid searches are merged and out of it the best parameter settings were chosen. The hard- and software specifications of evaluation computers are listed in appendix A.3.

Tracking on the CLUST testset was performed with python scripts and the RF implementation of scikit-learn. For this a classifier $RandomForestClassifier$ with the appropriate values for $n\_estimators$, $criterion$ and 3 as constant seed for the random number generator $random\_state$ were set. All other parameters were the scikit-learn defaults. The class probability value is retrieved from the RF $preditc\_proba$ function. For both, the correlation coefficient in the rough tracker's initial run and as part of the fine tracker, the OpenCV implementation $matchTemplate$ with TM_CCORR_NORMED was used. In order to define l_{thresh}, the highest class probability value of the 3 independent classifiers form the first estimation is used for RT_{Score}. The threshold l_{thresh} for significance testing from chapter 2.2.4.2 then is set to $l_{thresh} = \frac{2}{3}\ RT_{Score}$. For all sequences, 3 RF are trained, with independent pattern extraction areas from forest to forest (see the tables 10-12 in appendix A.2).

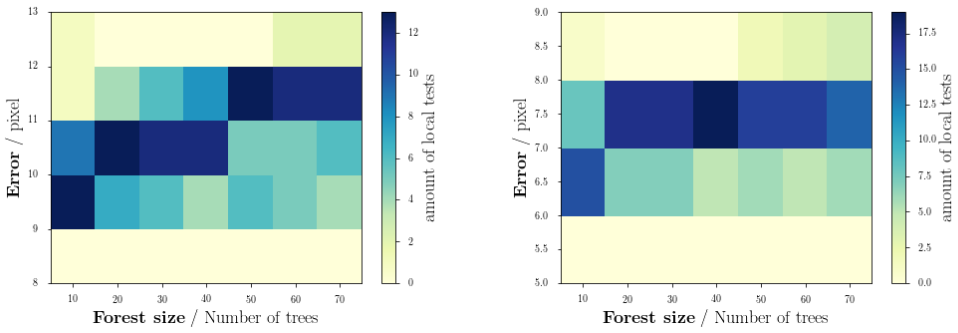
4.4.1 Trainingset evaluation

Starting with the algorithm's first part, the rough tracker. The parameter with a direct influence on its performance are the splitting criterion and dimension of the RF, the size of the ultrasonic patterns and the range of the landmark merger. Options like shifting the trajectory are first applicable when the fine tracker is engaged, but before inspecting the fine tracker, a robust input signal should be guaranteed to obtain adequate results. Therefore these options can be skipped at this first stage.

The figures 34 - 37 show different results, who are based on the same sequence of tests. Because many options are repeated with different hyper-parameters, some of the graphs are illustrated as heatmap to give a visual impression, especially about the center of a certain error distribution.

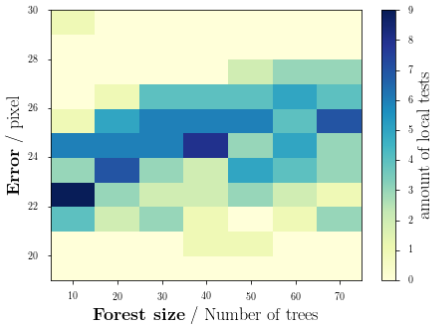
When inspecting the figure 34, a) and b) show for each of the evaluated forest dimensions a similar minimal error, where c) points out that especially the amount of 40 and 50 have achieved a better result than the others. However inspecting the particular error centers (illustrated by the color, where a darker color indicates a higher accumulation), clearly forests consisting of 10 decision trees achieved the minimal error for most of the cases. Whereas the criterion doesn't show a distinctive winner of the gini-entropy comparison

in figure 35, which coincides with the Elkan's opinion that there is no big difference to be expected, when using different criteria [44].



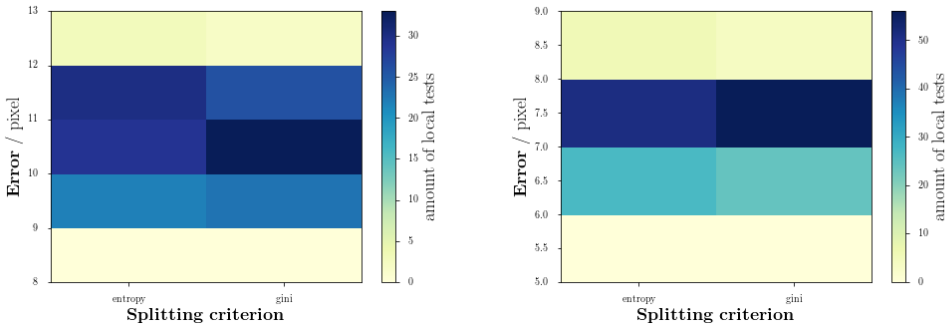
(a) Mean

(b) Standard deviation



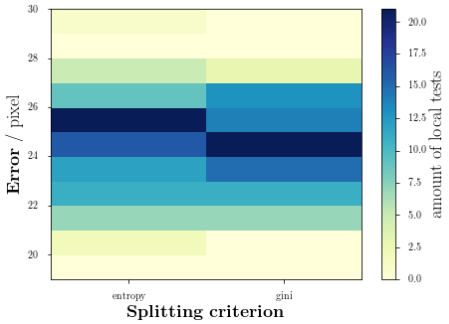
(c) 95^{th} percentile

Figure 34: The influence of the amount RF-trees



(a) Mean

(b) Standard deviation



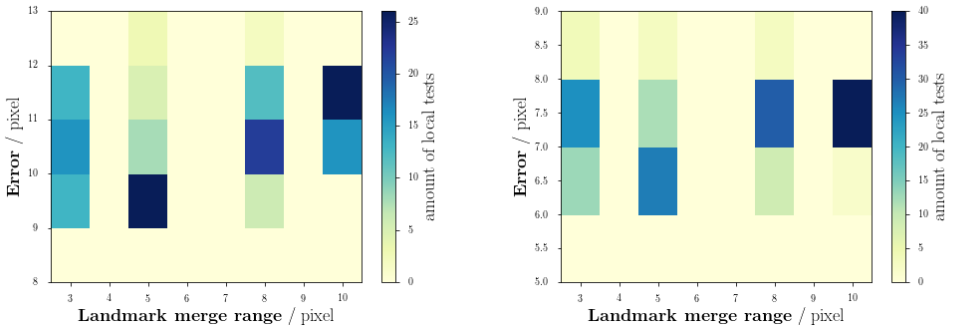
(c) 95^{th} percentile

Figure 35: The influence of the splitting criterion.

The next two parameter are directly responsible for the design and the size of the rough tracker's training database. While for a good generalization often a big dataset with a minimal amount of labels is admired, a higher range of the landmark merger is retrieving more data for less classes. In other words, if the number of different labels (i.e. the landmarks) is small, then automatically the merger's range increases, that combined retrieve more feature-vectors for a label and thus better results are expected to be acquired. The figures in 36 show the best amount of labels is 5, between the decision of extracting a big dataset and losing precision by getting a worse trajectory approximation (e.g.: 2 landmarks only describe a straight line).

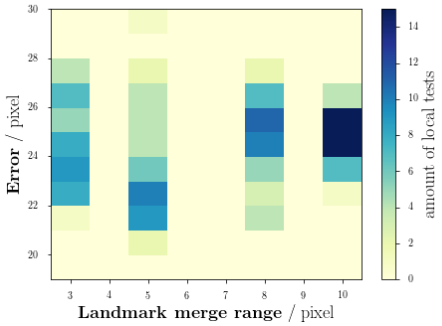
Where the landmark merger handles the labeled vectors, the pattern size manages the relative amount of features. Because speckle appearances are moving steadily, it needs to be inspected on the one hand, if more significant features can be extracted from horizontal or vertical oriented patterns. This way the preferred movement of the speckles should be assumed, with the sense that a certain feature is able to recognize a highlight

that was once seen before in another element. Then again, it needs to be considered, if small patches are sufficient enough for encoding the speckles or if a combined version (i.e. a square) is required. The results in the figures 37 show, the vertical patterns seem to retrieve more reliable results than the vertical opponents. Compared to the squared solution the difference is marginal, but the work load is increasing. Thus 50 pixel in height and 5 pixel in width for all patterns are set.



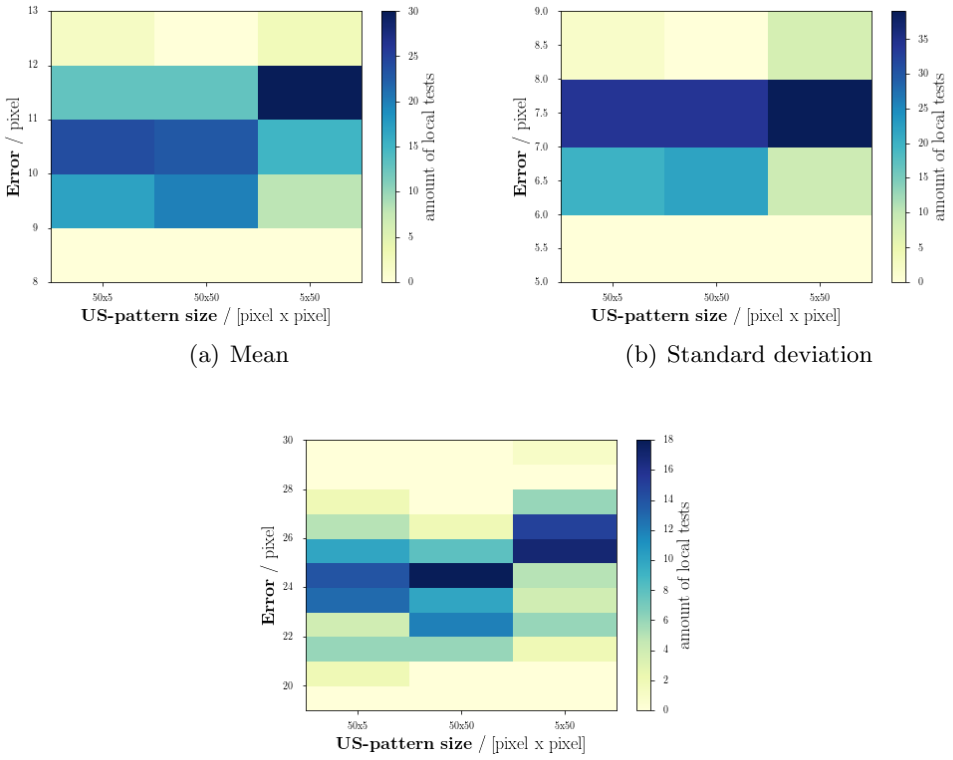
(a) Mean

(b) Standard deviation



(c) 95^{th} percentile

Figure 36: The influence of the merge range.



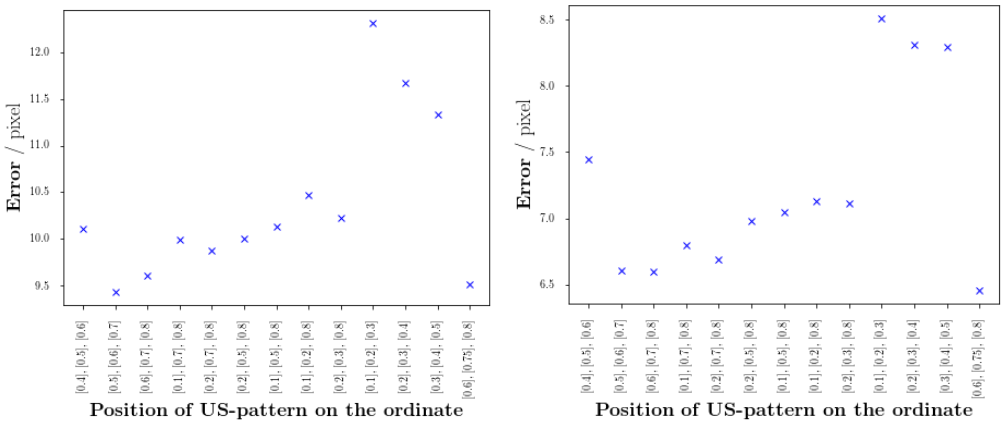
(a) Mean

(b) Standard deviation

(c) 95^{th} percentile

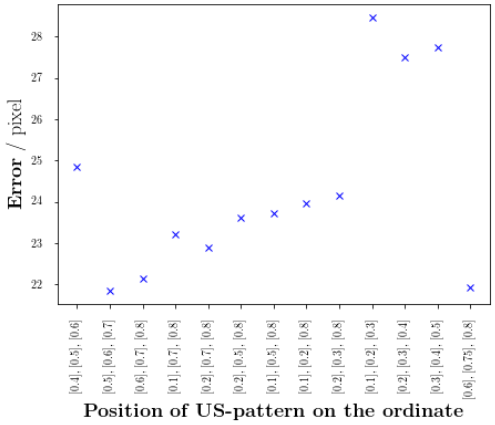
Figure 37: The influence of the ultrasonic pattern size

Certainly the localization and the amount of ultrasonic patterns take an important role too. Each position is relative to the dimensions of a full image, which varies from sequence to sequence. A pattern position is retrieved when (in this case) the image height is multiplied with one of the evaluated options (e.g. if the image height is 420 px and the option is 0.6 then the y-center of the pattern is at position 252). As described in chapter 3.2.3.2, using multiple, different combinations of the classifier's extraction areas are evaluated where the y-position of a single RF is denoted with square brackets in the figures 38. Here, the combination of independent RF [0.5], [0.6] and [0.7] have similar results with the sequence of classifiers at the positions [0.6], [0.75] and [0.8], where the later combination has a lower value in the standard deviation, which is a better indication for robustness than displaying the mean-error center. Because of this, it was selected for all later investigations. In any case, extracting the patterns from the lower half of the image yields lower errors.



(a) Mean

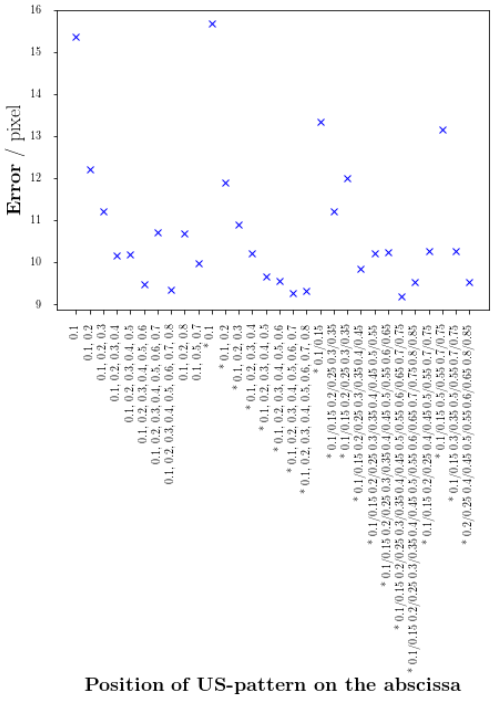
(b) Standard deviation



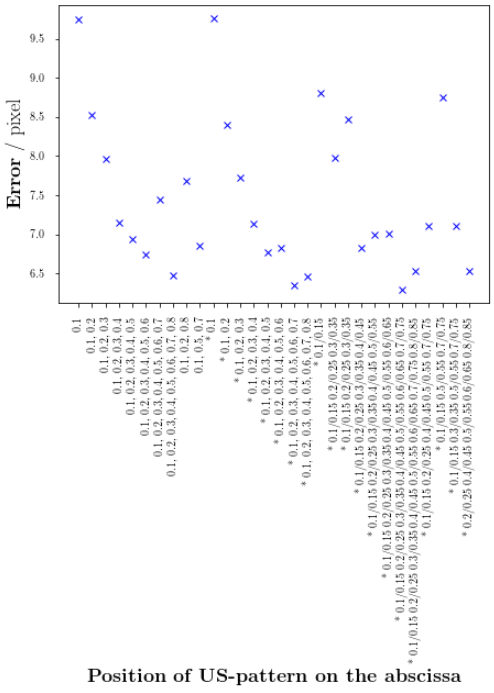
(c) 95^{th} percentile

Figure 38: The influence of the speckle pattern's y coordinate

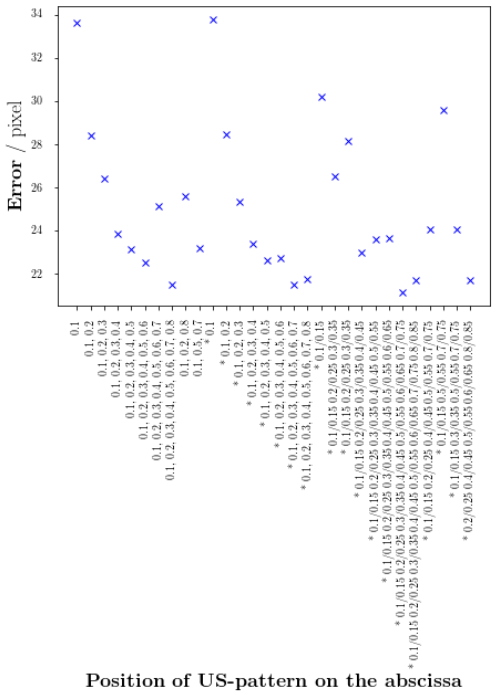
In the figures 39 the related x-components of the patterns are visualized, where 3 different tests are shown at once. The first simply links an option with each of the classifiers y-coordinate (e.g. with x-option 0.1, the pattern centers are [0.6, 0.1], [0.75, 0.1] and [0.8, 0.1], multiplied with the image dimensions respectively). The options marked with * have additional patterns, slightly shifted from the shown centers, with the idea of recognizing the speckle movement from one pattern into the adjacent patch again. The shift is -0.05 on the y-axis (i.e.: each classifier has now two areas on the ordinate to extract the features: [0.6, 0.55] [0.75, 0.7] and [0.8, 0.75]). The last approach is to provide individual y- and x- positions for all patches of the classifier. This is implemented in a manner that the -0.05 option of the y-coordinate gets combined with each x-coordinate behind the / in the figure 39 (see appendix A.2).



(a) Mean



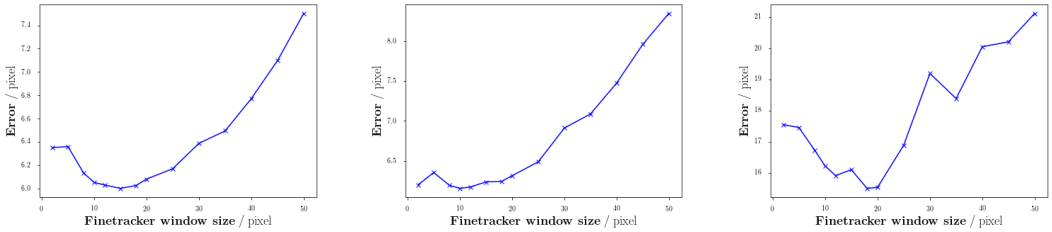
(b) Standard deviation



(c) 95^{th} percentile

Figure 39: The influence of the speckle pattern's x coordinate

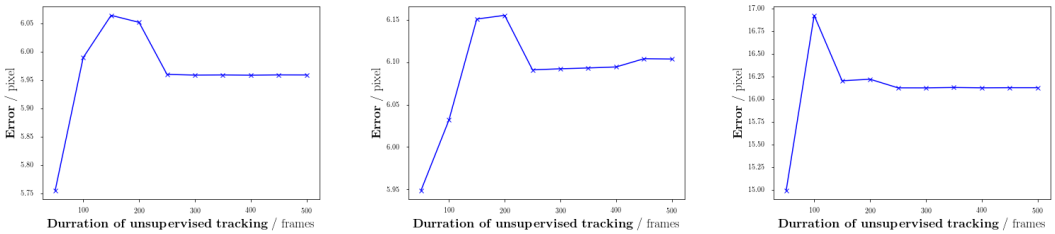
To engage the fine-tracker, the dimension of its search window F_s for retrieving the best results needs to be set. In the figures of 40, two minima at 10 and 18 stand out, where the later result obtains a lower error for the percentile but a higher in the standard deviation. The size of F_s then is calculated, when this value simply is added to both of the dimensions of F_r.



(a) Mean (b) Standard deviation (c) 95^{th} percentile

Figure 40: The influence of the dimension of the fine tracker's constricted area F_s.

After the relevant parts for the rough tracker have been handled, the question remains at which point a switch from the unsupervised tracking mode of the initial run, to the rough tracker's training, is appropriate. Even though the results in the figures 41 lead to a different assumption, to start training after the 50^{th} frame in particular, it is more recommended to start at the local minimum around 350, because 50 frames might not be enough in some particular cases for learning the basic trajectory. Besides the displayed error has a small range in general, which indicates that a higher amount of initial frames hardly take influence on the tracking performance.

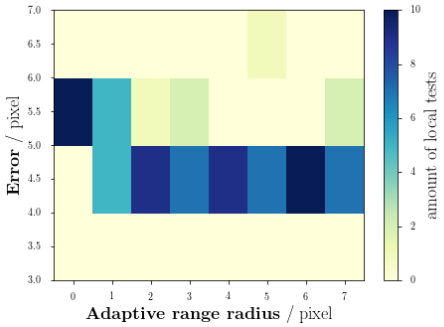


(a) Mean (b) Standard deviation (c) 95^{th} percentile

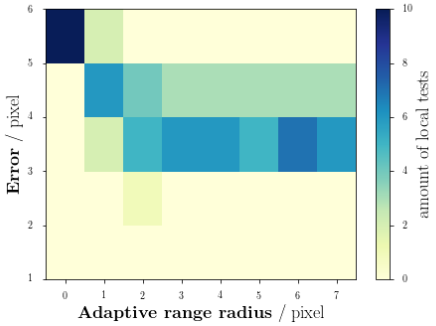
Figure 41: The influence of the duration of the inital unsupervised tracking.

Whenever the fine-tracker is enabled, the adaptive method and the frequency of forest

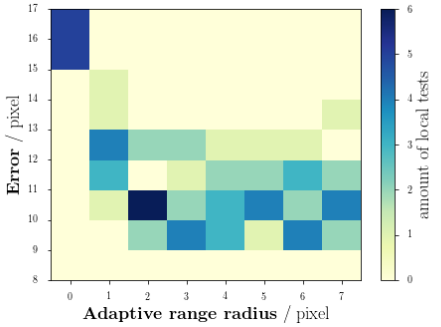
training can be evaluated. According to the figures 42 and 43, it is a decision between 2 or 6 pixel for the adaptive range and either retraining after achieving every 10^{th} or 20^{th} valid new sample.



(a) Mean

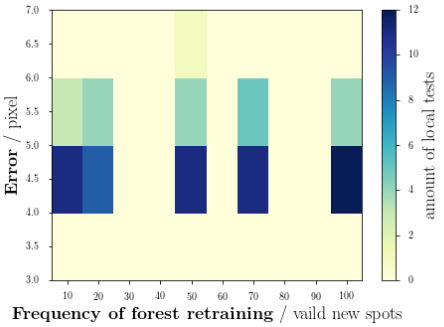


(b) Standard deviation

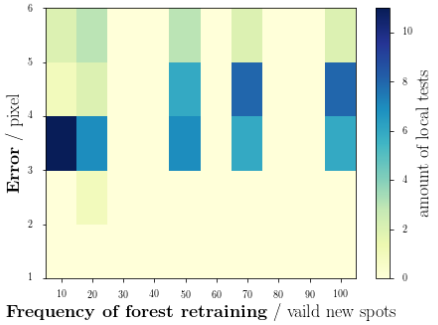


(c) 95^{th} percentile

Figure 42: The influence of the splitting criterion.



(a) Mean



(b) Standard deviation

(c) 95^{th} percentile

Figure 43: The influence of updating the RF.

Also choosing the parameter, that defines after how many frames the trajectory shall

be shifted, retrieves no clear result. Figure 44 leaves the impression that an appropriate selection is any value after 150 frames.

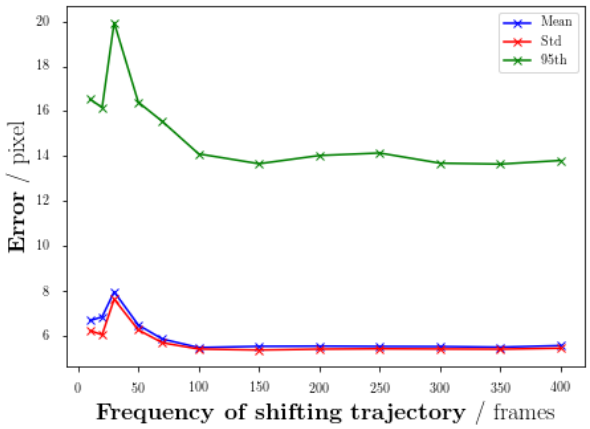


Figure 44: The influence of the trajectory-shift frequency.

To use a RF instead of the cross-correlation as a fine-tracker, seems promising to learn the significant parts of a ROI. Unfortunately, under all window sizes, the averaged mean tracking error for all sequences was always > 25 pixel, practically unusable. The actual problem is not the poor intermediate result, but the fact that this currently false predicted ROI position is used to extend its dataset, when its actually mandatory to learn from a true ROI center. Otherwise only a vague appearance gets learned over time.

Under all evaluated combinations of hyper-parameters, there is no configuration that clearly stands out against the others. Because of this, the different settings need to be joined and tested, to define the best model for evaluating the actual testset. The parameters, who are equally used for all of these final tests are: amount of trees (10), fine tracking method (correlation coefficient), duration of the unsupervised tracking $n_{init,max}$ (350 frames), ultrasonic pattern dimensions (50 pixel x 5 pixel), dimensions of the ROI reference window F_r (60 pixel x 60 pixel), relative pattern position on the abscissa (0.1/0.15 0.2/25 0.3/0.35 0.4/0.45 0.5/0.55 0.6/0.65 0.7/0.75), relative pattern position on the ordinate (1^{st} classifier: [0.55, 0.6], 2^{nd} classifier: [0.7, 0.75], 3^{rd} classifier: [0.75, 0.8]), adaptive range r_{ar} (2 pixel), amount of merged landmarks n_{lm} (5). Hence, the parameters that had no distinctive optimum are: splitting criterion (entropy and gini), search window size of the fine tracker F_s (70 and 78 pixel, equally for width and height), retrain frequency f_{rf} (10 and 20 valid new samples) and shift frequency $|F_{shift}|$ (250 and 350 frames).

After evaluating the mentioned combinations above, the best configuration is defined, whose standard deviation error was minimal, because this error type give a better im-

pression if the algorithm is performing reliably. The parameters without a distinctive optimum are set according to the configuration with the minimal standard deviation, as shown in table 7, whose results were for the trainingset (with an averaged factor of $\approx 0.4\frac{mm}{pixel}$, from the datasheet [6]):

mean tracking error	$\approx$ 4.261 pixel	$\approx$ 1.704 mm
standard deviation error	$\approx$ 4.024 pixel	$\approx$ 1.61 mm
95^{th} percentile	$\approx$ 10.8 pixel	$\approx$ 4,32 mm

Table 7: The final algorithm settings.

algorithm	Estimating landmarks
$\lvert I_{shift}\rvert$	250 frames
r_{ar}	2 pixel
n_{lm}	5
f_{rf}	20 $^{1}/_{valid\ new\ estimates}$
$n_{init,max}$	350 frames
dimensions of F_r	60 pixel x 60 pixel
dimensions of F_s	70 pixel x 70 pixel
dimensions of the patterns	50 pixel x 5 pixel
relative pattern positions	0.1/0.15 0.2/25 0.3/0.35
on the ordinate	0.4/0.45 0.5/0.55 0.6/0.65 0.7/0.75
relative pattern positions	1^{st} classifier: [0.55, 0.6],
on the abscissa	2^{nd} classifier: [0.7, 0.75], 3^{rd} classifier: [0.75, 0.8]
fine tracker method	correlation
criterion	entropy

4.4.2 Testset evaluation

For evaluating the actual testset from the CLUST challenge, the optimal parameters were chosen as mentioned in the previous section (see table 7).

Hardly information can be given about the runtime. Unfortunately it was not possible to solely execute the tracking algorithm on the computer. Moreover, multiple users have access to the system at the same time and thus it cannot be guaranteed that the algorithm could run on full performance, which certainly takes influence in the processing speed. However in table 8 a few selective examples are shown, that at least a rough impression can be given. Each ROI was evaluated sequentially. The average processing time per frame, for these examples range between $\approx$ 0.045 s to $\approx$ 0.248 s. Besides that the PC couldn't be used exclusively, there is another factor that takes influence on the timing results. Depending of the size from the rough tracker's dataset, it certainly takes more time to train the classifier, when the amount of feature vectors becomes higher. Because of the adaptive learning in the RF's testing phase, all datasets are increasing

continually, and enlarge their amount of data relative to the number of frames in the current sequence. There are multiple ways to eliminate this time problem, such as initially tracking all possible states of the trajectory and thus avoiding to adaptively extend the dataset and restarting the classifier's training, which is not feasible during the challenge environment.

Table 8: A selection of ROIs to give a rough impression about the runtime.

ROI	Amount of frames	processing time
CIL-03_1	1070	52 s
CIL-03_2	1070	72 s
ETH-06-2_1	5165	28 min, 53 s
ETH-10-2_1	5584	7 min, 27 s
ETH-10-2_2	5584	36 min, 6 s
ETH-10-2_3	5584	11 min, 12 s
ETH-12-1_1	14516	59 min, 1 s
ICR-09_1	3481	2 min, 38 s
ICR-09_2	3481	6 min, 4 s

After submitting the tracking series, the data was evaluated by CLUST and the received score is shown in table 9. This attributes the algorithm, great tracking qualities on arbitrary medical ultrasonic data. There are a few cases, where during the unsupervised stage the ROI couldn't steadily be recognized and thus a false trajectory was learned, that in the end yield in a general poor result. In a real scenario a support function for this stage could be added or simply the user monitors the short initial period and verifies, if the learned trajectory was correct.

Table 9: The summarized tracking errors of the testset.

Sequence	**Mean** / mm	**Std** / mm	**95th** / mm	**Min** / mm	**Max** / mm
CIL	1.7287	1.6055	4.8389	0.03836	13.8033
ETH	2.398	4.956	16.3749	0.0044	42.2414
ICR	1.6417	2.0684	4.7851	0.0107	21.4724
MED1	3.7332	7.3815	17.2261	0.0235	55.2858
MED2	1.6702	2.2358	4.8202	0.02705	28.5819
2D	2.4761	5.0861	15.1296	0.0044	55.2858

5

Conclusion

Tracking with random forest on medical ultrasonic image records is feasible in various ways. Many approaches have been implemented, that will be summarized below and also a few suggestions that may further increase either the algorithm's precision or robustness.

5.1 Summary

In this thesis two different concepts of utilizing RF as part of a tracking algorithm are presented. The idea of the one approach is to identify arbitrary pixels whether they are part of the object or not. Whereas the other solution estimates intermediate positions from a learned trajectory. Both have in common, that the algorithm gets started only with the initial position of a selected ROI. From there on, the training dataset needs to be generated, which is an individual process for both approaches.

For the classification, a small patch around the ROI is created, where different functions extract highlights and visual aspects, which are then used as the features. Exemplary functions are the intensity stretching, local mean values or a combination of two feature responses. Because only an initial annotation is provided, there is no particular information that could distinguish each pixel's group affiliation. Therefore these labels need to be generated artificially, in a way that each pixel with an intensity that approximates the ones from the center of the ROI and also its adjacent pixel are already identified as part of the target, should be considered as an object, otherwise it is part of the environment. The so generated dataset is then used to train the classifier, which starts estimating already on the next frame. Finally, when a sample image has been analyzed and the particular pixels have been estimated, the center of the object pixels can be retrieved (e.g. by calculating the centroid of the pixel mass, or the mean position value).

On many sequences the tracking with pixel classification could successfully be engaged. It is assumed that the basic kind of extracted features for the dataset, lead to a poor classification in a couple of cases, where the tracker completely looses the object. A better set of features might help to prevent these issues. However, then it is still unclear if such problems will reoccur in future, which is a sign that it might be not suitable for the medical field.

The landmark estimation consists of two main components. The first part retrieves a coarse, but robust position estimation upon a learned trajectory, the second one further analyzes this constricted area to retrieve the exact ROI position. With the algorithm's ignition, in a preceding unsupervised but short tracking procedure, the object's trajectory is estimated. During this period the intermediate positions are gathered, where in the mean time, ultrasonic patterns from different but static positions of the full image are extracted. These patterns are influenced from all elements, which are involved in the making of an ultrasonic scanning. Because the appearances of these patterns are reproducible, they appear to be capable of linking a pattern with a particular intermediate position on the learned trajectory. Since the trajectory varies over time, only a rough localization can be provided by this method. It is then the task of the fine tracker

to define the exact position within this constricted area. Additional support functions help to handle special tracking cases, such as an automatic trajectory shift, a landmark merger that artificially increases the training dataset, a non-ultrasonic image data interpolation and an adaptive trajectory adjustment that rearranges landmarks for the coarse estimation whenever the algorithm has learned enough valid new instances. All parameters are either defined by the algorithm itself (e.g. training dataset) or are figured out empirically during development (e.g. correlation window dimensions, or landmark merge radius). This is done in a way that for each algorithm invocation, only an arbitrary sequence and an initial position of the desired ROI needs to be provided and no information is transmitted between two sequences or annotations.

The tracking results of the landmark tracker were promising enough to prefer this variant over the pixel classification for further parameter tuning. This approach was also submitted to the CLUST2015 challenge, where it retrieved the following results for all 2D sequences: mean error ≈ 2.4761 mm, standard deviation error ≈ 5.0861 mm and the 95^{th} percentil ≈ 15.1296 mm.

General attributes of this variant are:

- Any element is traceable (e.g. tissues, highlights, arbitrary signals), if the fine tracker is able to recognize the ROI.
- No preparation required, could be applied at anytime.
- The highest error that occurs, if the trajectory was learned successfully, is the distance between both of the trajectory's ends including the related search areas of the fine tracker.
- A short initial warm up phase of a few breathing cycles, to generate the rough tracker's dataset
- Practically independent from previous results, because of the pattern analysis (i.e. the frames could be processed in an arbitrary order and don't need to be consecutive, when the algorithm is trained and the usual scenario is considered without growing trajectory).
- Achieves a high precision already with only a template matching for the fine tracking.
- Could be implemented, to be applied in real-time scenarios.
- Due to the nature of landmark estimation, strong differences for the ROI position statement between only two frames can possibly occur.
- Relies on the fine tracking method, for generating its dataset and also for retrieving the exact position.

With this algorithm, a user is able to track tissues on ultrasonic images, with minimal effort, as only the starting point of the desired ROI is necessary and in return, the related object's coordinates per frame are retrieved. Performance in real-time could only be indicated, because the most time consuming part is the forest training over the time, which shouldn't even be necessary when the user initially records all possible positions of the trajectory, just once. Hence, the initial requirements are mainly met.

5.2 Outlook

There are multiple ways of expanding or improving the algorithm, if the current state of the algorithm does not meet the requirements. Below a few promising options are briefly described.

First of all, there is the option to keep the algorithm as it is and simply replace the RF with another technique out of ML. A promising alternative is the SVM.

5.2.1 Pixel classification

More significant features are required to improve the pixel classification. Following the example of pattern recognition with CNN (e.g. handwriting, faces), specific features for each ultrasonic scanner could be obtained by learning every annotated ROI in the CLUST training set with the help of neural networks. Many different filter functions could then be copied and simply be integrated in the random forest's feature extraction.

Similar to the landmark estimation, during an initial unsupervised tracking, the trajectory could be learned. If this trajectory is then combined with a maximum distance to the estimated ROI as tolerance, then the tracker could be prevented from loosing the object.

5.2.2 Landmark estimation

In order to improve the exact localization, it might help to inspect further distance metrics such as briefly discussed in chapter 2.2.3 as an alternative for the fine-tracking algorithm. It might turn out to significantly increase the accuracy, when combining multiple metrics (e.g. correlation-coefficient and canberra-distance) to one algorithm in the sense of a sensor fusion.

The current state of adaptive learning could be further optimized, to achieve higher processing speeds. For the CLUST challenge it is designed in such a way that after a certain frequency the forest gets deleted and retrained (with the new data). This could be improved with an incremental induction, where the classifier simply gets edited, as mentioned in [81]. Another approach without further developing the algorithm, is to simply instruct the user to force every extreme trajectory position. This way the whole trajectory gets learned and updating the classifier once in a while is not necessary anymore.

Handling the algorithm's outliers could increase likewise precision and robustness. They could be suppressed, when during each iteration, subsequently to the tracker a filter is added. Implementations like finite and infinite impulse response (FIR, IIR respectively) are based on results from previous frames, thus whenever an outlier occurs, the error gets adjusted to the tracking history. However, in case of a system freeze such a detected jump is valid and needs to be excluded. To distinguish the validity, simply the timestamps of the frames could be compared, because this would also lower the current frame rate. Unfortunately this information was not provided by the challenge, thus this option was skipped.

Instead of pattern matching, the pixel classification could be applied as fine-tracker. This way the user combines the aspects of both algorithms. With the resulting method the user for instance would have the freedom of choice for defining the ROI center and also preventing the tracker to unintentionally move away from the trajectory.

References

[1] G. Reischl. Dr. 'da vinci' im op-saal der zukunft. http://futurezone.at/science/dr-da-vinci-im-op-saal-der-zukunft/24.574.612 (15.12 2016.).

[2] T. N. Payne F. R. Dautrive. A comparison of total laparoscopic hysterectomy to robotically assisted hysterectomy: Surgical outcomes in a community practice. *Journal of Minimally Invasive Gynecology*, 2008.

[3] European Cyberknife Center Munich. Fragen + antworten: Technologie, 2015. https://www.cyber-knife.net/de/patienten/fragen-antworten/detail/technologie.html (25.12.2016).

[4] Curexo Technology Corporation. About the robodoc® surgical system & benefits of robodoc, 2013. http://www.robodoc.com/learn/index.html#about (25.12.2016).

[5] Inc. Intuitive Surgical. The da vinci® surgical system, 2016. http://www.davincisurgery.com/da-vinci-surgery/da-vinci-surgical-system/ (25.12.2016).

[6] Valeria De Luca, Emma Harris, Muyinatu A. Lediju Bell, and Christine Tanner, editors. *Challenge on Liver Ultrasound Tracking*. CLUST ETH Zurich, 2015.

[7] M. Makhinya O. Goksel. Challenge on liver ultrasound tracking. *Motion tracking in 2D ultrasound using vessel models and robust optic-flow*, 2015.

[8] S. Kondo. Challenge on liver ultrasound tracking. *Liver ultrasound tracking using kernelized correlation filter with adaptive window size selection*, 2015.

[9] D. Nouri A. Rothberg. Challenge on liver ultrasound tracking. *Liver ultrasound tracking using a learned distance metric*, 2015.

[10] A. Hallack B. W. Papiez A. Cifor M. J. Gooding J. A. Schnabel. Robust liver ultrasound tracking using dense distinctive image features. *Challenge on Liver Ultrasound Tracking*, 2015.

[11] Z. Yaniv E. Wilson D. Lindisch K. Cleary. Electromagnetic tracking in the clinical environment. *American Association of Physicists in Medicine*, 2009.

[12] T. J. Mason J. P. Lorimer. *Applied Sonochemistry: Uses of Power Ultrasound in Chemistry and Processing*. Wiley-VCH Verlag GmbH & Co. KGaA, 2002.

[13] H. Amiri F. Benzarti. Speckle noise reduction in medical ultrasound images. *International Journal of Computer Science Issues, Vol 9, Issue 2, No 3*, 2012.

[14] G.R.Suresh S.Sudha and R.Sukanesh. Speckle noise reduction in ultrasound images by wavelet thresholding based on weighted variance. *International Journal of Computer Theory and Engineering, Vol. 1, No. 1, April 2009*, 2009.

[15] H.J. Tiziani. *Laser speckle*. http://dx.doi.org/10.18419/opus-4400, 1993.

[16] T. J. Mason J. P. Lorimer. *The laser guidebook*. R. E. Fischer W. J. Smith, 1992.

[17] A. Purde. *Speckle-Interferometrie zur Formvermessung unstetiger Oberflächen*. PhD thesis, TU München, 2005.

[18] R. W. Prager A. H. Gee L. H. Berman G. M. Treece. Speckle detection in ultrasound images using first order statistics. *CUED/F-INFENG/TR 415*, 2001.

[19] B. S. Carmo R. W. Prager A. H. Gee L. H. Berman. Speckle detection for 3d ultrasound. *Ultrasonics 40*, 2002.

[20] Murtaza Ali, Dave Magee, and Udayan Dasgupta. Signal processing overview of ultrasound systems for medical imaging. *SPRAB12, Texas Instruments, Texas*, 2008.

[21] Jkrieger on German Wikipedia 22:21 5.3.2007 CET Public domain via Wikimedia Commons. Laser_speckle_rot.jpg.

[22] E. Alpaydin. *Maschinelles Lernen*. Oldenburg Wissenschaftsverlag GmbH, 2008.

[23] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A Zadeh. *Feature extraction: foundations and applications*, volume 207. Springer, 2008.

[24] Aarti Singh. Practical issues in machine learning overfitting overfitting and model selection and model selection. *Machine Learning*, 10(701/15):781, 2010.

[25] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2-3):103–130, 1997.

[26] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

[27] K. Knight E. Rich. *Artificial Intelligence*. McGraw-Hill, Inc., 1991.

[28] R. Gerlach. *Reinforcement Learning: Roboternavigation in Heimumgebungen*. Diplomica Verlag Hamburg, 2012.

[29] Sun Yuan Kung. *Kernel methods and machine learning*. Cambridge University Press, 2014.

[30] M. Ester H.-P. Kriegel J. Sander X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 1996.

[31] Wen Zhu, Nancy Zeng, Ning Wang, et al. Sensitivity, specificity, accuracy, associated confidence interval and roc analysis with practical sas implementations. *NESUG proceedings: health care and life sciences, Baltimore, Maryland*, pages 1–9, 2010.

[32] Hans-Otto Georgii. *Stochastik: Einführung in die Wahrscheinlichkeitstheorie und Statistik.*, volume 4., überarbeitete und erw. Aufl of *De Gruyter Lehrbuch*. De Gruyter, 2009.

[33] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766, 2013.

[34] M. Imdadullah. Percentiles: Measure of relative standing of an observation within data. `http://itfeature.com/statistics/percentiles-are-measure-of-relative-standing-of-an-observation-within-a-data` (25.02.2017).

[35] Distance metrics. `https://numerics.mathdotnet.com/distance.html` (25.02.2017).

[36] Gutierrez-Osuna R. Intelligent sensor systems, course notes, department of computer science, wright state university. `http://research.cs.tamu.edu/prism/lectures/iss/iss_l9.pdf` (04.04.2017).

[37] scikit-learn developers. sklearn.svm.svc. `http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC` (25.12.2016).

[38] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[39] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[40] J Ross Quinlan et al. Bagging, boosting, and c4. 5. In *AAAI/IAAI, Vol. 1*, pages 725–730, 1996.

[41] Chen Jin, Luo De-Lin, and Mu Fen-Xiang. An improved id3 decision tree algorithm. In *Computer Science & Education, 2009. ICCSE'09. 4th International Conference on*, pages 127–130. IEEE, 2009.

[42] Wenliang Du and Zhijun Zhan. Building decision tree classifier on private data. In *Proceedings of the IEEE international conference on Privacy, security and data mining-Volume 14*, pages 1–8. Australian Computer Society, Inc., 2002.

[43] Stephenie C Lemon, Jason Roy, Melissa A Clark, Peter D Friedmann, and William Rakowski. Classification and regression tree analysis in public health: methodological review and comparison with logistic regression. *Annals of behavioral medicine*, 26(3):172–181, 2003.

[44] Charles Elkan. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, volume 17, pages 973–978. Lawrence Erlbaum Associates Ltd, 2001.

[45] Learn by marketing: Will. Decision tree flavors: Gini index and information gain. http://www.learnbymarketing.com/481/decision-tree-flavors-gini-info-gain/ (22.03.2017).

[46] César Ferri, Peter Flach, and José Hernández-Orallo. Learning decision trees using the area under the roc curve. In *ICML*, volume 2, pages 139–146, 2002.

[47] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[48] Wray Buntine and Tim Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8(1):75–85, 1992.

[49] D. D. Patil V. M. Wadhai J. A. Gokhale. Evaluation of decision tree pruning algorithms for complexity and classification accuracy. *International Journal of Computer Applications (0975 –8887) Vol II - No. 2, Dec 2010*, 2010.

[50] Jason Bell. *Working with Decision Trees*. John Wiley & Sons, Inc, 2014.

[51] Pere Torres, David Riaño, and Joan Albert López-Vallverdú. Inducing decision trees from medical decision processes. In *International Workshop on Knowledge Representation for Health Care*, pages 40–55. Springer, 2010.

[52] Ying Liu, Dengsheng Zhang, and Guojun Lu. Region-based image retrieval with high-level semantics using decision tree learning. *Pattern Recognition*, 41(8):2554–2570, 2008.

[53] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[54] Thomas G Dietterich. Ensemble learning. *The handbook of brain theory and neural networks*, 2:110–125, 2002.

[55] Robert E Schapire, Yoav Freund, Peter Bartlett, Wee Sun Lee, et al. Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5):1651–1686, 1998.

[56] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[57] Thomas G Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157, 2000.

[58] Tin Kam Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

[59] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[60] L Breiman. Manual–setting up, using, and understanding random forests v4. 0. 2003 http://oz. berkeley. edu/users/breiman. *Using_random_forests_v4. 0. pdf.*

[61] Dimitrios Ioannou, Walter Huda, and Andrew F Laine. Circle recognition through a 2d hough transform and radius histogramming. *Image and vision computing*, 17(1):15–26, 1999.

[62] scikit-learn developers. Choosing the right estimator. `http://scikit-learn.org/stable/tutorial/machine_learning_map/` (10.02.2017).

[63] Edgar Osuna, Robert Freund, and Federico Girosi. Support vector machines: Training and applications. 1997.

[64] Tao Li, Shenghuo Zhu, and Mitsunori Ogihara. Using discriminant analysis for multi-class classification: an experimental investigation. *Knowledge and information systems*, 10(4):453–472, 2006.

[65] Dijun Luo, Chris HQ Ding, and Heng Huang. Linear discriminant analysis: New formulations and overfit analysis. In *AAAI*, 2011.

[66] Peter Sollich and Anders Krogh. Learning with ensembles: How overfitting can be useful. *Advances in neural information processing systems*, pages 190–196, 1996.

[67] Pamela K Douglas, Sam Harris, Alan Yuille, and Mark S Cohen. Performance comparison of machine learning algorithms and number of independent components used in fmri decoding of belief vs. disbelief. *Neuroimage*, 56(2):544–553, 2011.

[68] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.

[69] Saeed Abu-Nimeh, Dario Nappa, Xinlei Wang, and Suku Nair. A comparison of machine learning techniques for phishing detection. In *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, pages 60–69. ACM, 2007.

[70] Justus Piater and Louis Wehenkel. A comparison of generic machine learning algorithms for image classification raphaiel marwee, pierre geurts, giorgio visimberga. 2003.

[71] Alexander Statnikov, Lily Wang, and Constantin F Aliferis. A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification. *BMC bioinformatics*, 9(1):319, 2008.

[72] Antonio Criminisi and Jamie Shotton. *Decision forests for computer vision and medical image analysis*. Springer Science & Business Media, 2013.

[73] Szilard Pafka. Benchmarking random forest implementations. `http://datascience.la/benchmarking-random-forest-implementations/` (11.02.2017).

[74] Erin leDell. Benchmarking random forest classification. `http://www.wise.io/tech/benchmarking-random-forest-part-1` (11.02.2017).

[75] W. G. Kropatsch H. Bischof. *Digital Image Analysis - Selected Techniques and Applications*. Springer-Verlag New York, Inc., 2001.

[76] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[77] Wilhelm Burger and Mark James Burge. *Digitale Bildverarbeitung: Eine algorithmische Einführung mit Java*. Springer-Verlag, 2006.

[78] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

[79] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[80] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[81] Paul E Utgoff. Incremental induction of decision trees. *Machine learning*, 4(2):161–186, 1989.

Appendix

A.1 Tracking error graphs of the pixel classification

This section contains a few selected examples to give an impression about the accuracy of the pixel classification. The images are provided from a different ultrasonic scanner on each of the results shown in the figures below.

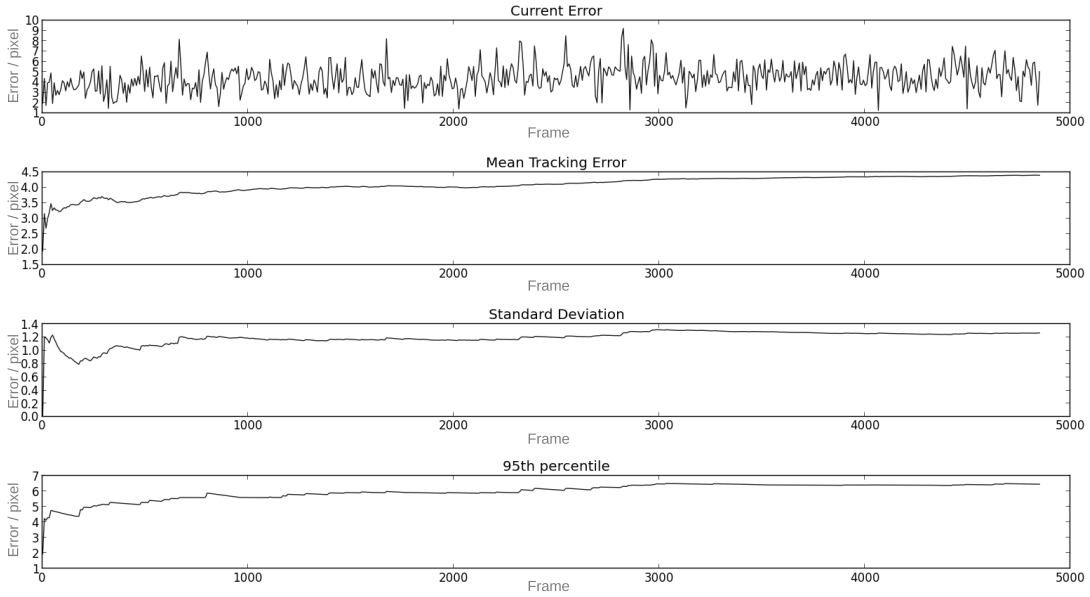


Figure 45: Tracking result of ICR-01_2.

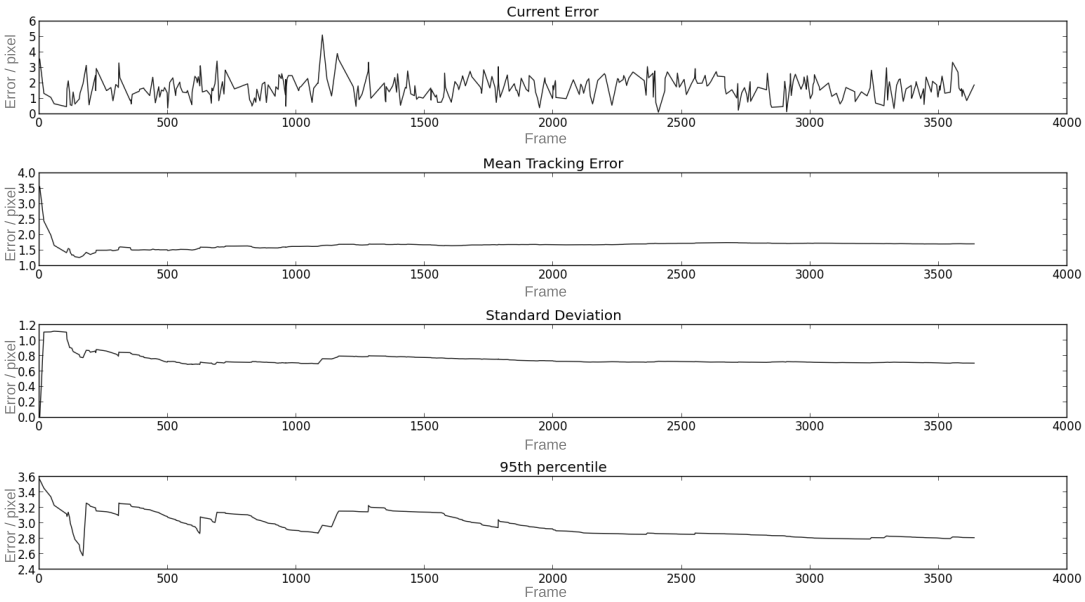


Figure 46: Tracking result of ETH-01-1_1.

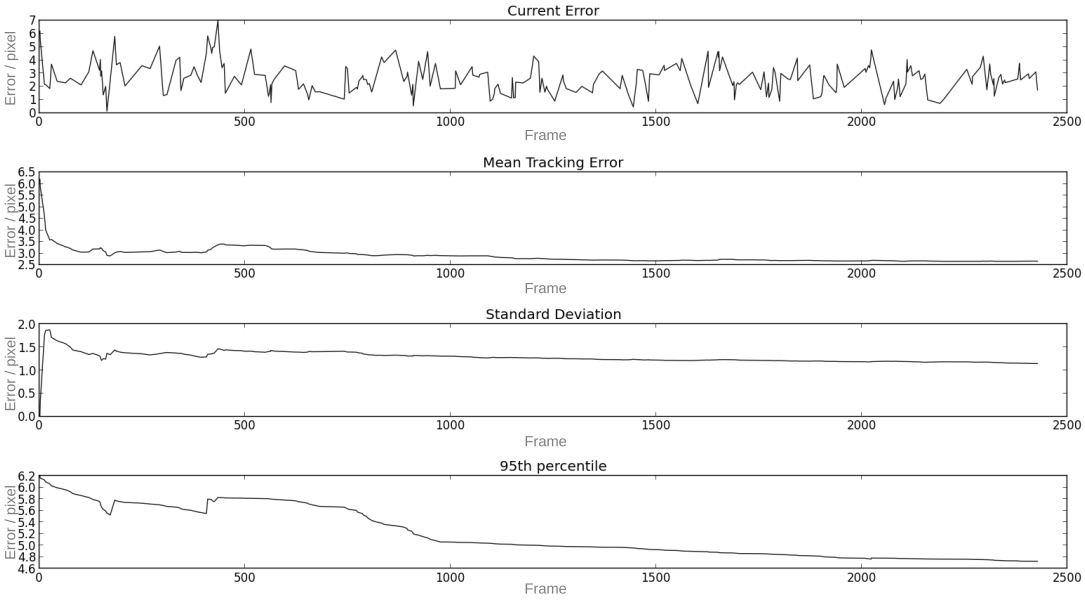


Figure 47: Tracking result of MED-02-1_1.

A.2 The design of the rough tracker's dataset

This section describes the rough tracker's dataset design, that is assembled online for tracking each ROI of the CLUST evaluation testset.

The number of patterns n_p for the rough tracker is 14 for one classifier, where each pattern's height p_h is 50 pixel and width p_w is 5 pixel. Since 3 independent RF were used on every ROI, the total amount of different patches then is 42, whose locations are relative to the full ultrasonic image dimensions of each sequence. The center positions for all 3 classifiers are shown in the tables 10, 11 and 12, with the units as percentage of the height and width from the entire image, for the y and x coordinates respectively (e.g.: if a frame's height is 240 pixels and the width 640, then, as an example, the center of the top pattern from table 10 is located at coordinate y=144, x=64).

Table 10: Patches for the first RF.

coordinates / %	
y	x
60	10
55	15
60	20
55	25
60	30
55	35
60	40
55	45
60	50
55	55
60	60
55	65
60	70
55	75

Table 11: Patches for the second RF.

coordinates / %	
y	x
75	10
70	15
75	20
70	25
75	30
70	35
75	40
70	45
75	50
70	55
75	60
70	65
75	70
70	75

Table 12: Patches for the third RF.

coordinates / %	
y	x
80	10
75	15
80	20
75	25
80	30
75	35
80	40
75	45
80	50
75	55
80	60
75	65
80	70
75	75

Every cell of a feature vector ν represents an unprocessed pixel-intensity extraction, out of the n_p patches from ultrasonic images. The individual pattern-pixels are accessed column by column, starting with the upper left, ending with the lower right element. The patterns in turn, are specifically arranged in the tables 10-12, with the same processing order from top to bottom, for the feature vectors. More precisely, when for instance for the first RF a ν is assembled, then the first feature is the intensity level of the top left pixel from the top pattern of table 10, whereas the last feature is the intensity level of

the bottom right pixel from the last pattern in this table.

For calculating the feature vector's length, the amount of color channels n_c for a pixel needs to be defined, which is 1 due to every image was handled as grayscale. Finally the length for one v and thus the amount of features per label can be calculated with equation 16.

$$|\nu| = label + n_c \cdot p_h \cdot p_w \cdot n_p = 1 + 1 \cdot 50 \cdot 5 \cdot 14 = 3501 \tag{16}$$

With the help of the image vector F_{init}, that contains the instances from the initial unsupervised tracking procedure, the individual features for each frame of F_{init} are extracted from the static pattern locations (see tables 10-12) and are then combined with the related ROI positions on the trajectory, saved during the unsupervised tracking period. This creates the initial dataset, which then is a $|F_{init}|$ x $|\nu|$ matrix for one RF.

A.3 System specifications

The system's specifications of the computers, who were utilized for tracking on the CLUST testset is shown in table 13. During evaluation of the trainingset, the group of computers had to be switched, that lead to slightly different results. The information in the table refers to the new group of PCs. Unfortunately information about the former devises could not be gathered anymore.

Table 13: The tracking system specifications.

Python	2.7.3
scikit-learn	0.11
OpenCV	3.1.0-dev
OS	Debian GNU/Linux 7.11
Kernel	Linux 3.2.0-4-amd64
GPU	Nvidia Geforce Gt 610
CPU	Amd FX-8350 8 x 4000 MHz
Memory	31.4 GiB

A.4 Hyper-parameter optimization

Table 14: The results of the tests, when the old group of computers is compared with the new.

PC	Trees	criterion	Compare method	Start training / Frames	Correlation Window size / px	Correlation / [px x px]	Speckle patch / [px x px]	X Percentage	Y Percentage	Retrain after / frames	Adaptive range / px	Shift after / frames	Mergedifference / px	Fine Tracking	Shift Trajectory	Adaptive Method	Handle Black Area	(avg) MTE / px	(avg) STD / px	(avg) 95th / px	Time / s
New	10	entropy	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.6624146577	6.7873262205	22.5792852229	3503.96605515
New	10	entropy	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	9.4144771565	6.665311038	22.4598369219	2445.44611502
New	10	entropy	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	10.8219571672	7.4275896131	24.94076886	2037.81603312
New	10	entropy	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.4124974386	6.576082939	21.8480056193	9285.64335608
New	10	entropy	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	10.3923598374	7.2172762878	24.2462927472	7957.12560606
New	10	entropy	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	9.6447980809	6.3739522936	21.2956330241	7422.35126495
New	10	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.5752184024	6.864661666	22.3672347787	3311.82590914
New	10	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	12.4496526886	8.6610895131	29.1215601381	2297.59667897
New	10	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	9.9532395808	6.7525549	22.0511419577	1958.27326202
New	10	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	10	FALSE	FALSE	FALSE	FALSE	11.0001142132	7.5939456688	25.3452324033	1805.56948781
New	10	gini	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.928147602	7.249908197	24.0826325085	3448.44288111
New	10	gini	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	9.3074581879	6.5780473718	21.732854993	2444.35873008
New	10	gini	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	10.9112885093	7.453894973	24.9619189787	2108.19057918
New	10	gini	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.6198239184	6.7818403891	22.6956609084	9283.281039
New	10	gini	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	9.4872832034	6.7496439102	22.3190016678	8200.13484097
New	10	gini	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	9.766639472	6.4257216844	21.5606791398	7285.46636415
New	10	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.7463553182	6.9790993182	23.422485623	3300.67582893
New	10	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	9.6570445614	6.9249167211	22.8418664509	2319.5854029?
New	10	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	10.1392260744	6.8946291936	22.6164856643	1926.78513193
New	10	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	10	FALSE	FALSE	FALSE	FALSE	10.6855093669	7.1925256343	24.2778049981	1857.55653715
New	70	entropy	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	10.2597874909	6.9962106906	23.472688332	8074.19203687
New	70	entropy	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	9.181084771	6.1827978514	21.0371117868	4836.12947798
New	70	entropy	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	12.2772913611	8.0941925819	27.5286622719	4063.22451997
New	70	entropy	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	11.0553373117	7.610436929	25.3107054	19506.169594
New	70	entropy	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	10.0727654306	6.911307301	23.42730681	14022.7109571
New	70	entropy	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	11.2183901502	7.4705364776	25.1731328293	11863.9820409
New	70	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	11.8600933712	8.198608518	27.7198082806	7716.85464382
New	70	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	11.7104221126	7.9428204298	26.7043826463	4656.05177093
New	70	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	10.4386580603	7.1825227468	24.0305372916	3892.65590596
New	70	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	10	FALSE	FALSE	FALSE	FALSE	11.5101236621	7.6256384814	25.8478736441	3625.34217906
New	70	gini	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	10.4976129234	7.2967993175	24.4510947304	7669.92705107
New	70	gini	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	9.3193865561	6.3464674867	21.1308533645	4810.68979788
New	70	gini	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	11.7141220106	7.8685132526	26.9134995195	4064.3694539?
New	70	gini	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.883284959	6.8245395314	22.8311921048	19131.518225
New	70	gini	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	9.3397197798	6.3418916273	21.4503851001	13622.735044
New	70	gini	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	11.2946098588	7.5874202689	25.8762013464	11435.650701
New	70	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	10.511370903	7.4480685159	25.0126128898	7497.34144592
New	70	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	12.046032388	8.2608222701	27.6569047306	4627.70747304
New	70	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	11.7810891263	8.0554113961	26.8592833336	3709.128757
New	70	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	10	FALSE	FALSE	FALSE	FALSE	11.8822128433	7.7980820641	26.4629231401	3641.23382497
Old	10	entropy	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	10.6490729293	7.7098674055	25.5677904358	26136.2111862
Old	10	entropy	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	8.8168306171	6.2731999545	20.675062288	16222.4704878
Old	10	entropy	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	9.6909176875	6.6682319488	22.4634083946	33658.5330288
Old	10	entropy	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.0506901836	6.4710552218	21.4073748979	35027.4618659
Old	10	entropy	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	12.0675971227	8.4332683984	28.5741228742	24246.7797189
Old	10	entropy	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	10.3391570488	7.220154975	24.4594702126	23366.0308309
Old	10	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.3585304955	6.7676621943	23.2632980603	6782.9842758174
Old	10	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	11.5985415363	8.4682926247	28.3858051354	5891.71122384
Old	10	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	10.4032927856	7.2998406852	24.4988168618	5290.8074185868
Old	10	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	10	FALSE	FALSE	FALSE	FALSE	10.6193526258	7.4940868599	25.0190574253	5194.9129369273
Old	10	gini	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.3239954509	6.879481424	22.4049177898	12690.741123
Old	10	gini	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	8.8168306171	6.2731999545	20.675062288	16222.4704878
Old	10	gini	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	10.6941372607	7.3553037102	24.6945453906	8271.32837296
Old	10	gini	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.3783431848	6.7931745788	22.4623050221	23558.3633251
Old	10	gini	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	8.8259676788	6.3044718143	21.3376781427	24255.7580638
Old	10	gini	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	10.431435739?	7.1765574082	23.9255344743	13671.265343
Old	10	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.4822274875	7.0359657993	23.4988641582	6716.7952718754
Old	10	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	10.4782633945	7.6756142117	25.488103745	5944.4168510414
Old	10	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	9.9154999691	6.9344481369	22.6913566304	5416.9543139947
Old	10	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	10	FALSE	FALSE	FALSE	FALSE	10.0498571641	6.805228458	22.3753353345	5215.7564585196
Old	70	entropy	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.675570827	6.7331265948	22.5157645416	50754.0462739
Old	70	entropy	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	8.8237517484	6.050454224?	20.58374213	19049.4269481
Old	70	entropy	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	10.7725158563	7.3236513212	24.9452891729	36485.795214?
Old	70	entropy	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	10.8085532825	7.5873069521	25.1504788106	58430.469491
Old	70	entropy	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	10.4798632884	7.1355468943	24.299945518	45035.258271
Old	70	entropy	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	11.4718807593	7.8923423052	26.9751269047	51799.5499461
Old	70	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.5455707595	6.877247049	23.2894128657	11283.9523835184
Old	70	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	13.0314312169	8.6497732128	29.2899038555	8056.8993628
Old	70	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	9.747390897	6.7378368989	22.1717363083	7218.5492692
Old	70	entropy	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	10	FALSE	FALSE	FALSE	FALSE	11.2522370876	7.5863042972	25.7607107868	7064.66599798
Old	70	gini	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	11.0586898124	7.8121405?2	26.0499858?14	30582.7581451
Old	70	gini	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	8.9261864507	6.1237617065	20.6020232718	68494.3268101
Old	70	gini	Correlation	200	2	60x60	50x5	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	10.6331561897	7.3063881141	24.8078421557	64600.080646
Old	70	gini	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	10.197742374	7.2748919863	24.5210955514	27718.7232141
Old	70	gini	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	9.5252762638	6.6203676433	22.3917505266	53807.99137
Old	70	gini	Correlation	200	2	60x60	50x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	11.0333429399	7.4966057024	25.4002407719	15108.8706149
Old	70	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	3	FALSE	FALSE	FALSE	FALSE	9.9429227629	7.2216031218	24.2982376724	11078.0361645214
Old	70	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	5	FALSE	FALSE	FALSE	FALSE	10.9758643393	7.7219088774	25.5340040239	8388.3305323129
Old	70	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	8	FALSE	FALSE	FALSE	FALSE	9.9154999691	6.9344481369	22.6913566304	7339.74061203
Old	70	gini	Correlation	200	2	60x60	5x50	0.2/0.25 0.4/0.45 0.5/0.55 0.6/0.65 0.8/0.85	0.6, 0.75, 0.8	20	3	100	10	FALSE	FALSE	FALSE	FALSE	11.2612019168	7.5394604998	25.3173828578	7220.72965002