UNIVERSITEIT·STELLENBOSCH·UNIVERSITY
jou kennisvennoot • your knowledge partner

# Automatic landmark detection on Tsetse fly wing images

Terms of reference

(Biomathematics Honours Project Final Report)

Author

18564321 DS. Geldenhuys

Under the supervision of

Dr Marijn Hazelberg and Jeremy Bringham

## Plagiarism declaration

I have read and understand the Stellenbosch University Policy on Plagiarism and the definitions of plagiarism and self-plagiarism contained in the Policy [Plagiarism: The use of the ideas or material of others without acknowledgement, or the re-use of one's own previously evaluated or published material without acknowledgement or indication thereof (self-plagiarism or text-recycling)].

I also understand that direct translations are plagiarism, unless accompanied by an appropriate acknowledgement of the source. I also know that verbatim copy that has not been explicitly indicated as such is plagiarism.

I know that plagiarism is a punishable offence and may be referred to the University's Central Disciplinary Committee (CDC) who has the authority to expel me for such an offence.

I know that plagiarism is harmful for the academic environment and that it has a negative impact on any profession.

Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully (acknowledged); further, all verbatim copies have been expressly indicated as such (e.g. through quotation marks) and the sources are cited fully.

I declare that, except where a source has been cited, the work contained in this assignment is my own work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

I declare that have not allowed, and will not allow, anyone to use my work (in paper, graphics, electronic, verbal or any other format) with the intention of passing it off as his/her own work.

I know that a mark of zero may be awarded to assignments with plagiarism and also that no opportunity be given to submit an improved assignment.

Signature: [illegible]

Name: Dylan Geldenhuys Student no: 18564321

Date: 20/10/2019

## Abstract

Experts in the field of Tsetse fly research have suggested that wing shape morphology can be used to determine some behaviours of this fly. This paper presents an approach to identify landmarks on Tsetse fly wing images that will be used to calculate wing measurements. The approach uses feature descriptors called Haar-like descriptors that are used as inputs to a tree-assemble machine-learning model. The model is trained using labelled images in a supervised manner. The algorithm presented in this work proves effective in accurately determining such landmarks, but fails to identify landmarks on some images.

## Acknowledgements

The author of this paper wishes to acknowledge Professor John Hargrove and his teams for acquiring the necessary wing data to carry out this study, as well as the Stellenbosch mathematics and applied mathematics department for creating the digital dataset and labelling the training set. Great thanks to Dr Marijn Hazelbag and Jeremy Bringham for the all support, supervision and guidance for this project. Finally, the author thanks to the mathematics department and SACEMA for providing the necessary funding for this paper.

# Table of contents

# List of figures

# List of tables

# 1 Introduction

## 1.1 Background

African *trypanosomiasis* also known as sleeping sickness is a disease transmitted through Tsetse flies (*Glossina spp*). Tsetse flies inhabit 36 countries in sub-Saharan Africa and act as a vector for *trypanosoma* the parasite responsible for sleeping sickness (Vale et al 2015). Sleeping sickness is a potentially fatal disease to humans, but it is also a major problem in cultivating livestock, as it is responsible for a disease named 'Nagana' in cattle. Nagana causes lethargy and lowered fertility that becomes widespread throughout a herd and often results in many deaths. This results in economic decline. To address this problem there have been numerous control strategies that have attempted to eradicate and control the Tsetse fly population in areas with livestock, but as soon as the control operation is over, the treated area is often re-populated by flies invading from adjacent untreated areas (Vale et al 2015). Under the leadership of Prof W. John Hargrove former director of SACEMA, a data set consisting of near 400,000 images of Tsetse fly (Glossina spp.) wings has been acquired. The digital image database was created from preserved Tsetse fly dissections collected over 15 years. Prof W. John Hargrove has proposed that analysing the wing shape variation with respect to season and long term changes could shed light on the migratory behaviours of the Tsetse fly populations, allowing us to typify Tsetse populations and identify them as a potential threat for re-invasion (Hargrove & Hazelbag 2019).

## 1.2 Objectives

The analysis referred to above requires some feature points on the wing to be defined that can be easily identified on all wings. Subsequently, the Cartesian coordinates of these points must be labelled on every wing in the data set. These feature points will be termed 'landmarks'. In this research, landmarks have been defined as the junctions of well-defined veins in the wing. With such a large number of wings to be labelled, this task cannot be carried out manually; instead, the aim is to develop a machine-learning algorithm that can be trained to recognize automatically landmarks on the Tsetse fly (Hargrove & Hazelbag 2019). In essence, the task is to identify landmark pixels in each image of the data set.

In this paper, a potential method for finding a single landmark on a Tsetse fly wing is provided as a proof of concept for reasonably identifying landmarks on a Tsetse fly wing images. This is first done with respect to this data set. In its application, it may be considered for deployment to detect all landmarks on the entire data set.

## 1.3 Motivation

If a technique that can reliably detect some landmark can be demonstrated, further development of this model to detect all landmarks in Tsetse fly wings will be possible, and finally deployment of this model for use on the entire set of unlabelled data. This data set can then be linked to other information that has been gathered about the Tsetse fly which includes seasonal, geographical and biological information. In effect, the results of this research could be used in developing an additional data set of morphological shape that could be added to the Tsetse fly database. This richer database could be used to ask further research questions about Tsetse fly behaviour and other environmental influences.

# 2 Literature review

The problem posed in this research, ‘finding all the landmarks on an image’ is well known in the field of computer vision and machine learning. The term 'landmark localization' is defined by the automatic localization of key points in an image (Ouanan H, Ouanan M & Aksasse, 2016).This is precisely the task that this research intends to accomplish on this data set.

## 2.1 Previous work on the Tsetse fly dataset

Previously in an Honours research paper by Josias (2017), the task of landmark detection on the same Tsetse fly wing data set was explored. He evaluated techniques using image processing followed by feature extraction which included Hugh transform, radon transform and Harris corner detection. These techniques did not rely on and learning algorithms and only used mathematical techniques used in computer vision. The results indicated that image processing techniques are able to identify intersections only in certain, optimal conditions. There are too many exceptional cases that must be addressed in order for the techniques to generalise well. The paper evaluated the performance of the methods using the Euclidean distance from the true landmark to the predicted landmark. The results of this evaluation (for landmark no.7) are presented below.

| | **Hough transform** | **Radon transform** | **Harris method** |
|---|---|---|---|
| **Landmark 7** | 31.6 | 11.9 | 4.3 |

*Table 1: Euclidean distance error for image processing techniques used in Josias (2017).*

Josias also tried using learning algorithms, namely, feed forward neural network and a convolution neural network. These learning algorithms performed fairly well achieving an accuracy of 98.8% for correctly identifying potential landmarks.

However, the algorithm was not designed to give a single landmark prediction, and thus the Euclidean distance error was not calculable.

## 2.2 Applications on biomedical images

In the biomedical field, there are a number of applications that involve the same task – automatically finding some landmark on bio-images. For instance, one can use landmark positions in X-ray images for the assessment of spinal flexibility and knee alignment angles (Wang et al. 2019).Since biological images are being dealt with, applications of landmark localization on biological images are considered.

The most successful landmark detection algorithms have made use of pixel classification or regression using machine learning methods followed by global landmark structure refinement (Ibragimov, Likar & Pernus, 2012; Donner et al 2013; Linder & Cootes 2015). The last step of global landmark refinement chooses the best points found by the machine learning method.

Machine learning algorithms that have been used in biological images have made use of tree-based ensemble machine learning methods (Criminisi & Shotton 2013). These algorithms have been employed in many image classification tasks (Marée, Geurts & Wehenkel, 2016 ;Wade et al. 2015). In the biomedical domain it has been used for organ segmentation (Lindner & Cootes 2015; Cuingnet et al. 2012), as well as landmark detection in radiology (Ibragimov, Likar & Pernus, 2012; Donner et al 2013; Linder & Cootes 2015).

In the past few years, there has been a great need to develop landmark detection algorithms for cephalometric images. The motivation is that the identification of cephalometric landmarks on images of the skull allows quantification and classification of anatomical abnormalities. In an effort to solve this problem, the ISBI cephalometric challenges in 2014 and 2015 compared landmark detection algorithms on a Celaphatoetry data set. The best performing algorithms were by (Donner et al. 2013) and (Lindner & Cootes 2015). These algorithms are briefly described below.

### 2.2.1 Ibragimov, Likar & Pernus (2012)

Ibrahimović combined random forests with a game theory tools that learn the relationships between landmark positions (global landmark structure refinement).

### 2.2.2 Lindner and Cootes (2015)

The algorithm in Linder and Cootes (2015) was developed for and applied to a Celaphatomtery data set. The algorithm is divided into two parts. First, individual landmark offset regressors are trained for each landmark. The offset regressors are used to build an active shape model, which gives an optimal configuration of landmarks through an iterative process – global landmark structure refinement.

Since biological images can be markedly different, a landmark detection algorithms performance will most likely vary for different data sets. In a recent paper by Vandaele et al. (2018) state-of-the-art landmark detection algorithms used for biological images were compared and tested on multiple data sets. These data sets consisted of 138 coloured images of Drosophila wings, 100 lateral human cephalometric radiographs and 113 ventral views of head skeleton of zebra fish larvae. The algorithms tested included the algorithms Linder and Cootes (2015) described above as well as Donner et al (2013). Ibrahimović et al Ibrahimović, Liker & Pernus (2012) algorithm was included only for comparison with its performance on the cephalometric dataset. The Donner et al algorithm is described below.

### 2.2.3 Donner et al (2013)

This can be described in the following steps. First, a random forest classifier is trained to classify pixels to create a probability map for each landmark. Thereafter, an offset regressor is trained to narrow the possible landmark pixels. Lastly, the final positions are chosen using a random field based on distances between the landmark positions.

### 2.2.4 Vandaele et al. (2018)

Vandaele et al. (2018) also introduces a new algorithm that is evaluated and compared to the above algorithms. It was designed to be used as a novel generic learning-based approach for landmark detection. This algorithm can described in three steps. In the first step, a separate extremely randomised forests classifier (ERFC) is trained to identify landmark pixels (for a specific landmark). In the second step, pixels are sampled from the general area of the landmark, whereby a multi-resolution approach is used to extract features for each pixel. In the third step, features are extracted from resized windows centred at the pixel coordinate. This algorithm identifies a few landmark candidates. The median landmark then chosen as the final prediction from each ERFC. The paper also did a systematic analysis on parameter values and feature descriptors used in the model and gives suggestions for method choices.

A summary of the main features of all the algorithms tested in Vandaele et al. (2018) is given below.

| | **(Vandaele et al., 2018)** | **(Lindner, C. & Cootes,2015)** | **(Donner, R. et al,2013)** | **(Ibragimov, B. et al, 2019)** |
|---|---|---|---|---|
| **Pixel descriptor** | Haar-like | Haar-like | Gaussian sub | Haar-like |
| **Learning model** | Extremely randomized trees (classifier) | Random forests (regression), PCA model | Extremely randomized trees (classification and regression), Markov Random Field | Random forests (classification), Gaussian model |

*Table 2: Summary of algorithms tested in Vandaele et al. (2018)*

The performance of these algorithms on the Droso data set will be most useful in considering the model used in this research since the Droso wing is comparable to the Tsetse fly wing. The results of this paper show that Vandaele et al. (2018)

algorithm performed the best on all three data sets. It was shown that Haar-like feature extraction performs the best on the Droso data set compared to the other feature extraction methods tested and the level of accuracy obtained using the best parameters and methods gave 29 994 being correctly classified out of 30 000 for a sampled image.

This paper implements and adapts the methods based on the findings and suggestions provided by Vandaele et al. (2018) to develop an algorithm that will effectively detect a specified landmark on a Tsetse fly wing.

# 3 Content chapter

## 3.1 Material and Methods

### 3.1.1 Materials

The Tsetse fly wings used in this research were from the species Glossina Pallidipes and G. M. Morsitans. The wings were collected over an eleven-year period, from Rekomitjie Research Station in Zambezi Valley of Zimbabwe. The physical data set consists of more than 150 000 pairs of wings. Part of this data set was then digitised by taking pictures of each fly wing using a handheld microscope camera. The resulting digital data set consists of over 10 000 images of Tsetse fly wing, pairs of size 1280x1024 pixels. Around 10 000 of these wing pairs were manually labelled and the coordinates of the landmarks were saved in a separate document. Of this data, 464 images were selected to be used in this research. Figure 1 shows an example of a Tsetse fly wing with all its landmarks plotted.

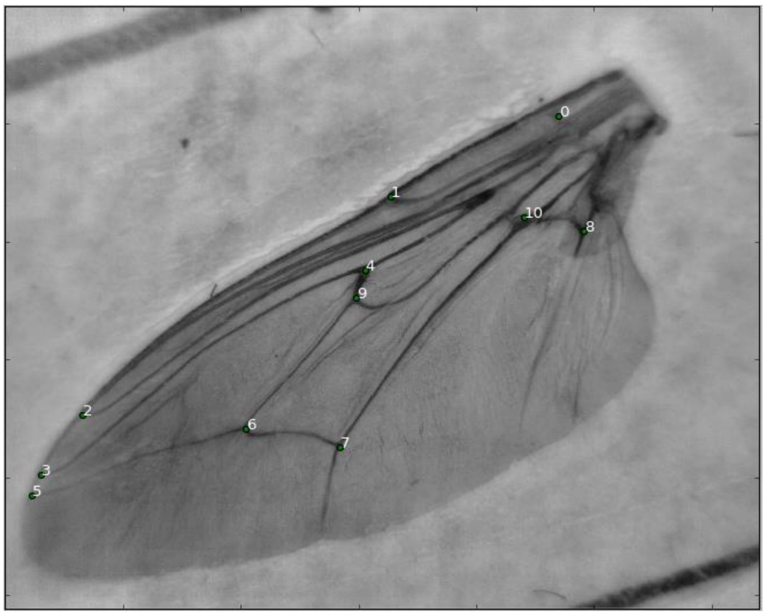


*Figure 1: Tsetse fly wing image with all its landmarks plotted*

In total, there are ten landmarks on a Tsetse fly wing. All of these landmarks are shown in figure 1 above.

### 3.1.2 Data pre-processing

Since the aim of this paper is only to prove that this methodology can work on this data set, only left wings and landmark 7 (seen in figure 1) were considered. For convenience, there is no distinction made between the landmarks or left and right wings in the following sections.

The original data chosen for this research consisted of a file containing images and a file containing the landmarks for each image. The data was cleaned by removing any images that contain broken wings or had been labelled incorrectly. To do this, every image was plotted with its landmark and manually checked to remove unwanted images. The reason for removing such images is to avoid including unnecessary noise into the training data.

### 3.1.3 Algorithm description

The algorithmic task is to be able to classify an observed pixel from an image as being a landmark or not. This problem was tackled using a supervised learning approach. This requires some input data features and an output label/ground truth to train on (model fitting). The features are used as inputs to make a classification and the label is then used as a reference to optimize towards a better classification during training. The algorithm is trained using pixels sampled around the landmark. Each pixel is given a label. Pixels in the neighbourhood of the landmark are labelled as 'landmark' and all others as 'non-landmark'. The trained model will then take pixels from an unseen image and make a binary classification, landmark or not. The positive landmark predictions are then aggregated to choose the final landmark position. The steps of this algorithm are explained fully in the following subsection.

#### 3.1.3.1 Extraction of pixels

Each observation is a point of the image that has coordinates (x,y) and corresponds to some pixel, each observed pixel is also given an output label. The pixels are described by a 287 feature descriptors. Below we show how we choose the output label for each observed pixel. In the proceeding sections 'pixel sampling scheme' and 'feature extraction' we show how pixels are sampled on the image and how features are extracted for each pixel.

#### 3.1.3.2 Labelling the data

Consider the coordinate of the landmark. Starting at this point, one can increase the size of this point as some disk until it is considered to be covering the complete landmark, or at least what the human eye considers to be the all of the landmark. The same idea is used to draw some radius around the true landmark whereby all coordinates inside this circle are considered a landmark, and all pixels outside this circle to be non-landmark pixels. To do this a parameter $\boldsymbol{R}$ was defined. if one obverses a landmark at position $(\mathrm{x}, \mathrm{y})$ the output label will be 'pos' if $(x - x_l)^2 + (y - y_l)^2 \leq \boldsymbol{R}^2$ , 'neg' otherwise, where $(x_l, y_l)$ is the true landmark coordinate. This method of labelling was inspired by Vandaele et al. (2018).

#### 3.1.3.3 Pixel sampling scheme

To train this model, one must sample pixels from each image and determine whether these pixels are landmarks or not. It is unfeasible to sample every pixel in an image, thus one must consider what pixels will be used from each image in our training set. It is possible to sample pixels uniformly, but this would produce an unbalanced training set. If one draws some radius around each landmark and calls all the pixels within this radius 'landmark pixels' and uniformly sample the image we would get far more negative landmarks than positive landmarks. For instance, if a radius of 9 pixels is chosen around the landmark then there would be 799 landmarks pixels and 1309921 non-landmark pixels in a single image. Uniformly sampling would then produce a 0.06: 99.94 ratio of landmark and non-landmark pixels samples. To create a more balanced pixel sample for each image we choose a radius of 9 pixels, and sample $\boldsymbol{\rho}$ pixels within the radius and $2\boldsymbol{\rho}$ pixels outside the radius $\boldsymbol{R}$. In general, it can be expected that the same landmark across the images is located within the same general area. If this is considered for the prediction stage, it is reasonable to only sample pixels within the same general area of the image to find the landmark. If this method of pixel sampling during prediction is used, Then it is best to constrain the training set to only consider pixels within some $\boldsymbol{R_{max}}$, where pixels within $\boldsymbol{R}$ are labeled as landmarks and pixels between $\boldsymbol{R}$ and $\boldsymbol{R_{max}}$ are labelled as non-landmarks.

#### 3.1.3.4 Feature extraction

For each pixel, the algorithm must be given some useful features as inputs that will allow it to classify the pixel correctly. To extract features for each pixel some window was cropped around the it. The steps taken in this method are as follows. First, for each pixel a window size of $(2\delta \times 2\delta)$ centred at the pixel coordinate is cropped. Next, the window resolution reduced by a factor F, where F is a method parameter. From each of these windows extract 287 feature descriptors. The feature descriptors chosen for this application are called Haar-like feature descriptors as suggested by Vandaele et al. (2018). The first and second steps are illustrated below in figure 2.

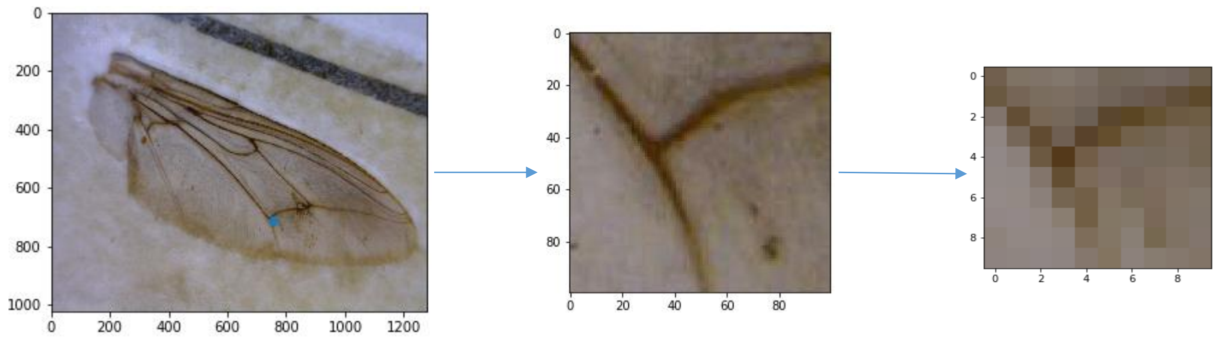


*Figure 2: Window extraction process*

### 3.1.3.5 Haar-like features

A Haar-like feature descriptor is a real valued number obtained by taking the difference in summed pixel intensity of neighbouring rectangle sections of the image. Each pixel will have its surroundings described by several Haar-like features, which will be the input values for the machine learning model. The effective use of Haar-like features was first introduced as one of the contributions in Viola and Jones (2001). Figure 3 below gives an example of a few Haar-like features taken from a reduced resolution image of a Tsetse fly wing.

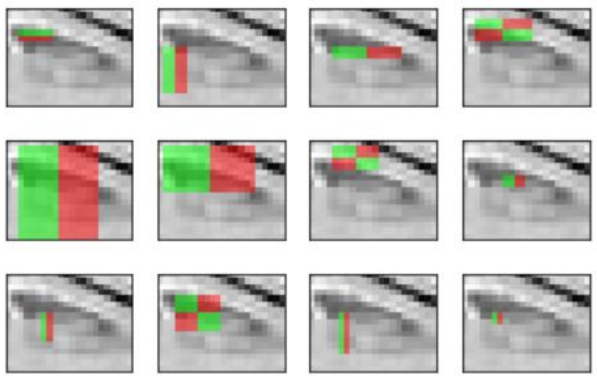


*Figure 3 : Haar-like feature example*

Since pixel intensity is the focal point here, each image is converted to grey scale to avoid having to deal with RGB values of the colour image. This way only the single grey scale pixel values are usd when calculating Haar-like features. Five different

Haar-like descriptor types are considered in this research. These Haar-like features are presented below.

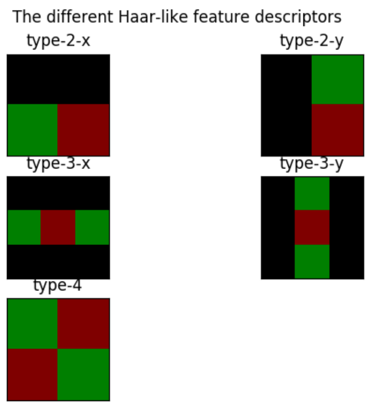


*Figure 4 : Haar-like decritor types used*

Calculating all Haar-like features is infeasible to do on a grey scale image. Instead, an integral representation of a grey scale image is used, similarly used by Viola and Jones (2001). This processes the image such that each 'pixel coordinate' now has an integral representation, such that any location (x,y) in the image is the sum of all the pixels to the left and above it including itself. This can be stated mathematically as,

$$ii(x,y) = \sum_{x' \leq x, y' \leq y}^{n} i(x', y') \qquad (1)$$

Using the integral image representation significantly reduces the computational time needed to calculate the Haar-like features of an image.

### 3.1.3.6 Machine learning model

The machine learning algorithm used in this research is a random forest classifier (RFC). A random forest is an ensemble of decision trees trained on different subsets of the training data. The decision trees give a vote for the classification. The random forest chooses the classification that gets the most votes. To understand the random forest model and the parameter values that are used in this research, it is necessary to understand the underlying algorithm for each decision tree.

A decision tree consists of multiple layers where each layer contains some nodes. This is where a branch is split on some decision. In this scenario, the split is done on some feature and the decision is whether the value for that feature is smaller or larger than some value. Based on the answer it will move further up the tree until it eventually reaches a 'leaf' which classifies the input. When a tree is fully trained, we expect these decisions to be able to classify the pixel based on its Haar-like features.

Training this tree involves choosing features and values on which to split. There are two main optimization procedures that use different metrics to choose the optimal splits, these are the Gini Index, and the Entropy function which uses information gain as a metric. This research uses the Gini Index.

Each node is given a Gini impurity; the probability that a randomly chosen sample in a node would be incorrectly labelled if it were labelled by the distribution of samples in the node.

The Gini impurity is described by the equation,

$$\boldsymbol{Gini = 1 - \sum_{i=1}^{C}(p_i)^2} \qquad (\mathbf{2})$$

Where $\boldsymbol{p_i}$ is the ratio of class $\boldsymbol{i}$ in the node whose Gini index is being calculated. There are a total of $\boldsymbol{c}$ classes, which in our case is binary $(\boldsymbol{c = 2})$. At each node, the decision tree searches for the best feature value to split on that reduces the Gini impurity the most. This is done recursively until it reaches a specified threshold for the Gini value. Effectively, the algorithm aims to minimise the Gini impurity at each node until it reaches a max depth with a sufficiently low Gini value.

A decision tree is very susceptible to over-fitting. Over-fitting occurs in the case of a very 'flexible model'. This happens when one does not limit the number of layers and the tree essentially memorises the data from which it learned. When this happens, the model is not only learning relationships but also learning the noise attributed to the training data. A model like this is said to have high variance because the model parameters will essentially change considerably given a different sample of the same data set. The opposite case were the model is inflexible is said to have a high bias where it over generalises.

A random forest solves this problem of variance versus bias by aggregating many decision trees and averaging their predictions. A random forest utilises two concepts to achieve this optimally; random sampling (with replacement ) of training data points when building trees, and random subsets of features considered when splitting nodes (Louppe et al 2014).

### 3.1.3.7 Landmark prediction

First, define the standard deviation and the average of the landmark pixels across the training images as $\boldsymbol{\mu}$ and $\boldsymbol{\sum}$. Proceed as follows to make landmark predictions on unseen images. First, randomly draw $\boldsymbol{N_p}$ pixels from a normal distribution $\boldsymbol{N}(\mu, \sum)$. Next, apply the trained random forest classifier on the sampled pixels from the unseen images. Lastly, the classifiers then output a number of positive landmarks, the landmarks are aggregated to find the median pixel coordinate. The median coordinate is used as the final landmark prediction.

### 3.1.4 Testing and validating

The algorithm was trained on 120 images. The algorithm was tested on 65 unseen images with $\boldsymbol{N_p} = 200$, and 35 with $\boldsymbol{N_p} = 1000$.

The final predicted landmarks given by the algorithm are compared to the actual landmark using the Euclidean distance between the final predicted landmark and the true landmark.

| **Parameter** | **Description** | **Value used** |
|---|---|---|
| $\boldsymbol{\rho}$ | the number of pixels sampled within R | **100** |
| $\boldsymbol{\delta}$ | half the length/width of the feature window | **50** |
| $\boldsymbol{F}$ | The factor that the widow resolution is reduced by | **10** |
| $\boldsymbol{R}$ | The distance to the landmark position determining the training pixel output class | **9** |
| $\boldsymbol{R_{max}}$ | the maximal distance to the interest point to extract non-landmark observations | **300** |
| $\boldsymbol{T}$ | The number of trees in the random forest | **10** |
| **Descriptor** | The feature descriptor used | **Haar-like** |
| $\boldsymbol{N_P}$ | Number of pixels sampled during prediction | **200, 1000** |

*Table 3: Description of main paramaters used in the algorthim*

### 3.1.5 Implementation

The algorithm was implemented using Python Scikit-image for Haar-like feature extraction and the Random Forest Classifier was implemented using the Scikit-learn library. Training the random forest classifier takes a few seconds. The specifications the computer were 8G ram i5 processor.

## 3. 2 Results

This section assesses use of this method to effectively identify landmarks on the Tsetse fly data set.

Of the 185 images used in this research 120 of them were used for training the random forest. The other 65 images where used for testing. The sampling and feature extraction part of the algorithm took considerably longer as the amount of pixels sampled for prediction was increased. To avoid the long run time, testing had to be split into multiple sections whereby we test using 200 followed by 1000 samples per an image

### 3.2.1 Testing $N_p = 200$

When testing using 200 samples, 65 unseen images were used to test on. Below is an example of an image and the pixels sampled from it. The coloured dots indicate the sample locations.

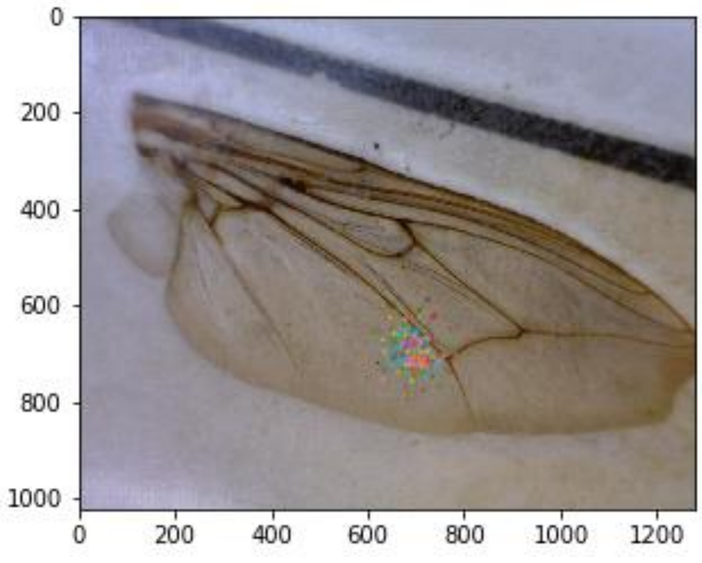


*Figure 5: pixel sampling scheme for* $N_p = 200$

Note that the sample region is offset from the landmark in this image. Since Gaussian distribution sampling was used, images with landmark coordinates that lie closer to the standard deviation of the data set will have less samples near its landmark.

Figure 6 below shows some of the landmark pixels identified by the algorithm on unseen images.

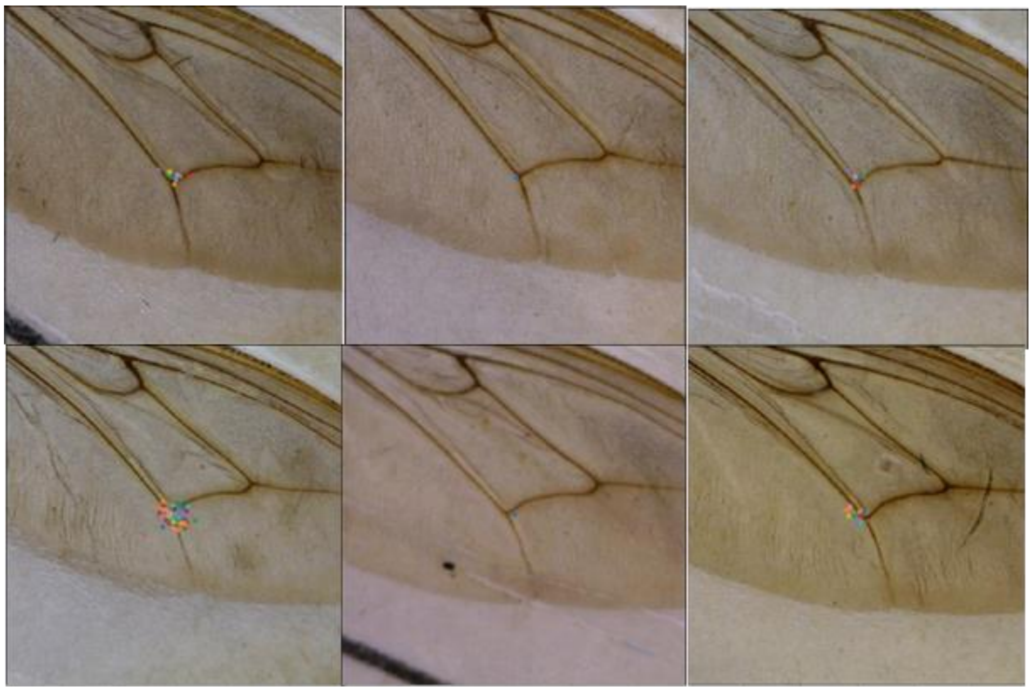


*Figure 6 : example of landmark pixels identified ($N_p = 200$ )*

Out of the 65 images that where tested only 47 of the images had at least one positive prediction made. The algorithm was unable to identify any landmark pixels on the rest of the test images. Note that some of the images contain only one positive landmark prediction (in blue) out of the 200 pixels sampled. Other images contain many landmark predictions.

To make the final prediction the landmarks shown above are aggregated and the median pixel value was chosen. The Euclidean distances from the final prediction to the true landmark label is given below.

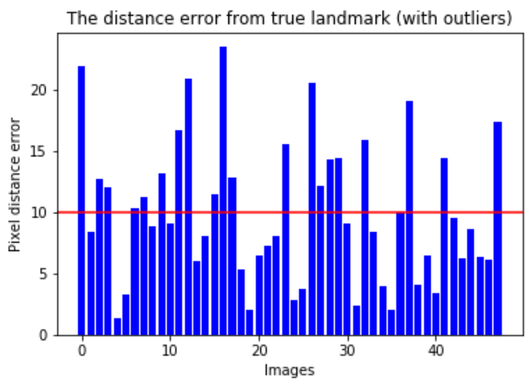


*Figure 7: Pixel distance errors (with outliers)*

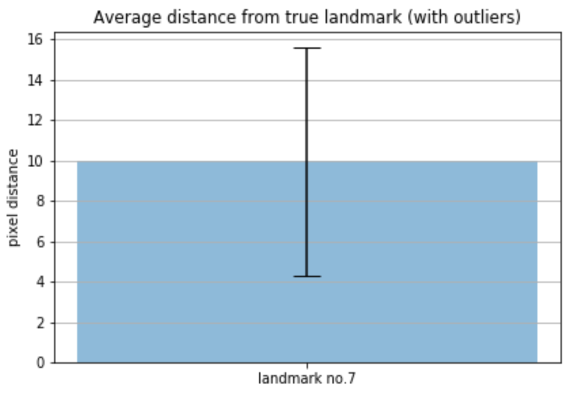


*Figure 8: Average pixel distance error and standard deviation (with outliers)*

The results above show that the mean pixel distance error was 9.1723 and the standard deviation was 5.2279. The standard deviation is high. This is due to some outliers that had a high distance error. Ideally, the algorithm will make predictions with a maximum error close to 9 pixels.

Subsequent to this test, all outliers that had a distance error of more than 12 were removed. The results are shown below in figures 9 and 10.

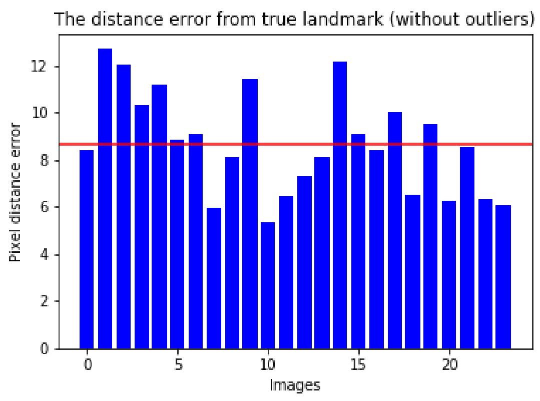


*Figure 9 : Pixel distance errors (without outliers)*

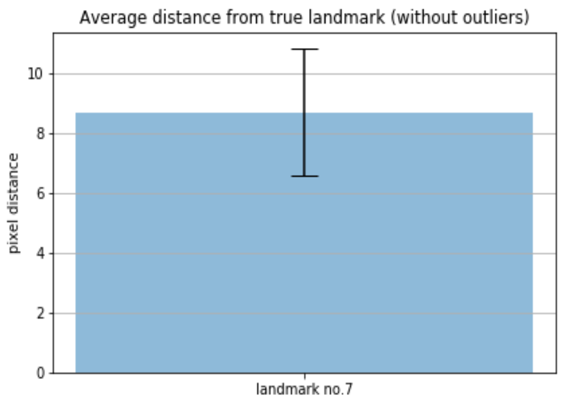


*Figure 10: Average pixel distance error and standard deviation (without outliers)*

When the outliers are removed there are only 24 images are left. The mean distance error is 8.727 and the standard deviation is 1.594. The mean distance error is only slightly decreased when outliers are excluded, but the standard deviation is reduced considerably as expected.

### 3.2.2 Testing $N_p = 1000$

When testing the algorithm using 1000 samples per an image (as suggested in the Vandaele et al. (2018)) Only 36 images were used. Of the 36 images, 33 of the images had at least one positive landmark classification. This is considerably better than when $\boldsymbol{N_p} = 200$. An example of the sampling scheme is presented below.

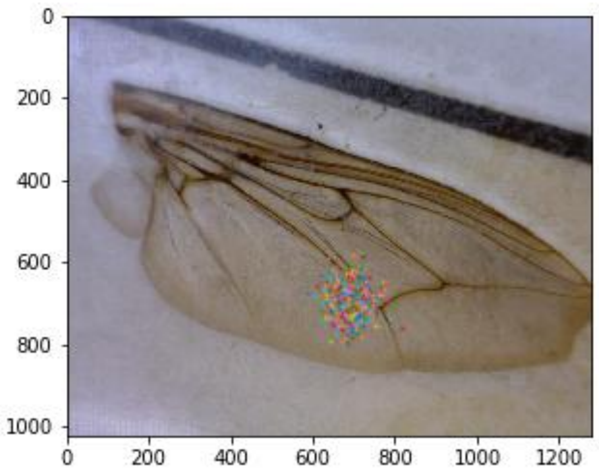


*Figure 11: Pixel sampling scheme for* $N_p = 1000$

Figure 12 shows some examples of the landmark predictions made by the algorithm for a few unseen images.

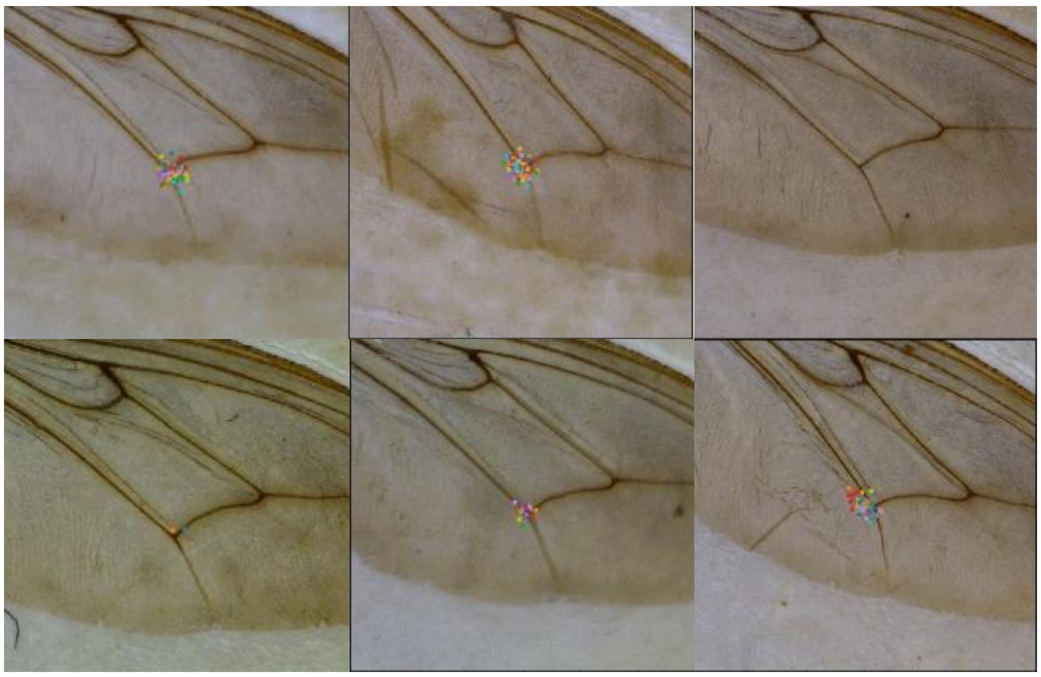


*Figure 12: example of landmark pixels identified* ($N_p = 200$ )

The number of landmark predictions are considerably more when using a 1000 sample scheme. Some of the images are still not classified well by the algorithm and only have one or a few landmark predictions. The results of the Euclidean distance error is shown given below in figures 13 and 14.

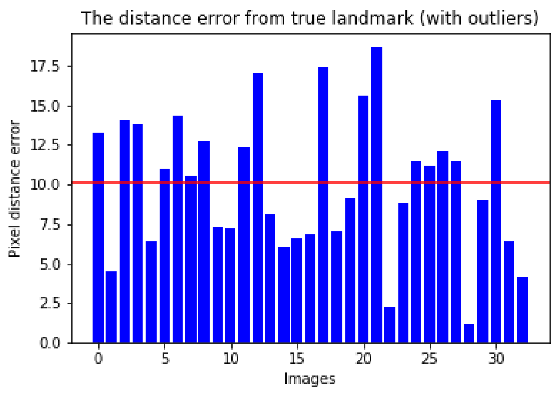


*Figure 13: Pixel distance errors (with outliers)*

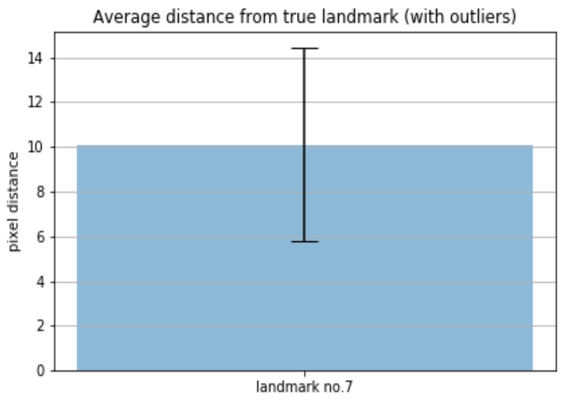


*Figure 14: Average pixel distance error and standard deviation (with outliers)*

The mean pixel distance error was 10.10, and the standard deviation was 4.35. This is not very different from the previous section where 200 pixels were sampled. Figure [..] indicates that there are a few outliers with large prediction errors. These outliers are less than the previous section but this may just be because the test set was smaller. Figures 15 and 16 present the results when outliers are removed.

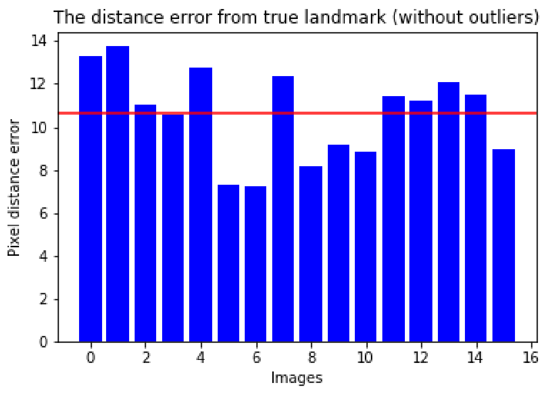


*Figure 15: Pixel distance errors (without outliers)*

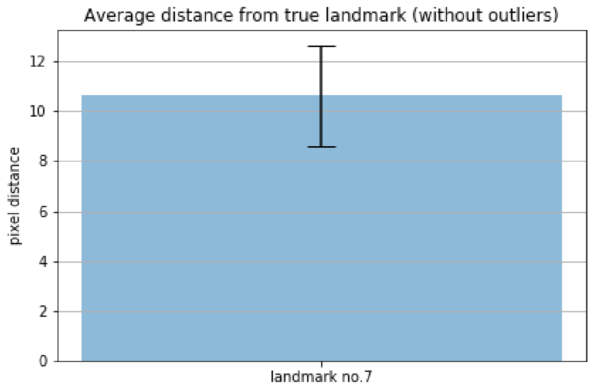


*Figure 16: Average pixel distance error and standard deviation (without outliers)*

If one removes outliers, only 16 of the 36 images are left. The mean distance error is 10.607, and the standard deviation is 2.02.

## 3. 3 Discussion

### 3.3.1 Model parameter choices

This section provides a brief discussion on parameter choices and how they may affect the algorithm.

The parameters used in this model were chosen by using the suggestions from Vandaele et al. (2018), and considering other models with similar performance. The window size was chosen such that for all pixels within radius $\boldsymbol{R}$, sufficiently contained a clear view of the intersection. The sampling strategy was chosen by considering the sampling scheme in Vandaele et al. (2018) for the Droso data set and choosing a similar positive (landmark) and negative (non-landmark) sampling ratio. The Haar-like feature descriptors chosen were the default ones given in the Scikit-image library. These Haar-like features were not specifically selected for, and their relative contribution in the decision trees of the rainforest is unknown. Further investigation would need to be done to decide on the optimal Haar-like feature descriptors and perhaps better Haar-like descriptors could be developed. During prediction, the sampling radius was chosen based on the standard deviation and a Gaussian distribution was used to sample. This was the same sampling strategy used in Vandaele et al. (2018). Using a Gaussian distribution means that outliers are less likely to be found or may have worse average prediction, instead it made be better to sample densely but randomly for unseen images. Sampling densely comes with a trade-off, reducing either the sample space or increase in computation cost.

For the parameters chosen during training, the negative sampling space is 1112 times larger than the positive space and only twice more pixels are sampled from the negative space. This means that the positive samples are densely collected and negative sampling area is more sparsely populated. This allows a better distinction between landmark pixels and non-landmark pixels.

### 3.3.2 Results discussion

The results of this paper show that, in general, the algorithm was not able to make predictions for all images but the algorithm performed well with respect to the mean pixel distance error in terms of accuracy and precision. The biggest concern is that that some images are not well classified; they contain none or few positive landmark predictions.

The difficulty in classifying some of the images may be due to the sampling window chosen. Instead of resizing the entire window, only a small window around each sample was resized. Another reason may be that number of Haar-like descriptors for each sample was not enough. In Vandaele et al. (2018). 1536 descriptors were used per a sample compared to 287 used in our implementation.

In all cases, the results show that some images had many landmark predictions made by the algorithm, whereas others had none or very few. It is not immediately obvious why this happened. Upon further inspection, the algorithm seemed not to perform well when the images where blurred. However, this was not always the case.

When testing the algorithm with different sample sizes, little variation in the accuracy and precision of the algorithm was observed. However, there was a large difference in the ability to classify all images. This may not be valid as the test sizes were almost half in size. The images contained in the smaller test set for $N_p = 1000$ may have been ones that the algorithm was able to classify more easily by chance. This research did not go into great depth about which select images were not classified by the algorithm. The most likely cause is due to the variation in the training set. The test set may only contain a few outliers in terms of landmark location, rotation and focus, and therefore would not be able to generalise well for these outliers. With some inspection of results, it seemed the algorithm struggled to classify blurred images. In general, this has been a problem in landmark detection studies especially in face reconition (Li et al. 2018).

In the first part of the results section, the algorithm was tested using only 200 samples per an image. From the 65 images tested only 47 where classified and when outliers were removed only 24 where left. This is considerably less than the initial test set.

Note that in the predication stage, the Euclidean distance from the true landmark for each image was averaged. To increase the accuracy and reduce the amount of classified images removed as outliers, one could remove outliers from the initial predictions made by the algorithm for each image. This would theoretically decrease the standard deviation and increase the amount of images classified after outliers are removed.

In the second stage of testing, when using 1000 pixels, a similar effect when removing outliers was observed. The increase in sample size did not seem to increase the accuracy of the algorithm, which is what was expected. The same method discussed above should decrease the amount of outliers.

### 3.3.3 Computational performance

The algorithm is fairly computationally inefficient compared to Vandaele et al. (2018). Sampling 1000 pixels per an image as suggested by Vandaele et al. (2018) took around 14 seconds. In Vandaele et al. (2018) it took less than a second. The random forest algorithm makes predictions almost immediately. Considering we have a total dataset of over 10,000 images the algorithm would need to be optimized significantly to be considered for deployment in labelling the full dataset. There are a number of ways the algorithm can be optimized. The algorithm in Vandaele et al. (2018) is far more efficient than the algorithm developed in this research. The inefficacy of the algorithm is most likely due to the data handling in the algorithm since the feature extraction methods should compare similarly to Vandaele et al (2018) as we used optimized built in functions from the scikit-image library.

### 3.3.4 Limitations

The findings of this paper are only applicable to one of the landmarks on the Tsetse fly wing. The landmark in this case was very well defined. For this reason, further testing would need to done on other landmarks. Another point to concider is that some images may not be classified well using this method. For instance blurred images.

## 3. 4 Conclusion

The aim of this research was to provide a methodology that could be used to identify landmark coordinates in a data set of Tsetse fly images. The use of machine learning algorithm has proven to be useful in similar tasks, and it has been proposed that using learning algorithm may provide a way of automatically completing this task. This paper used methodologies from current state-of-the-art algorithms for landmark detection to develop an algorithm that could identify a single landmark on the given data set of Tsetse fly wings. The results of this paper show that the algorithm performed well in terms of accuracy and precision but presented two main problems that would need to be investigated further before continuing to develop this model. The first problem is that the algorithm was never able to identify landmarks for all images in the test set. The second problem is that the computational efficiency of the algorithm was too slow to realistically label the entire dataset and would need to be optimized further. Additionally, another consideration is that the algorithm was only evaluated for one of the landmarks on Tsetse fly wing. The landmark that the algorithm was evaluated on is arguably better defined than some of the other landmarks in the wing, and training the algorithm for these landmarks may present new problems.

In going further with this research, the algorithm will need to be optimized sufficiently for it to be effective for the labelling the entire data set. The algorithm should always be able to predict some landmark given and image to a degree of accuracy of around 9 pixels. The algorithm would then need to be tested for other landmarks and be able to perform with similar results.

The method explored in this paper shows promise in being able to identify effectively landmarks on a given Tsetse fly wing image. The predictions were on average very accurate, but the algorithm failed to identify landmarks on some unseen images. The efficiency of the algorithm would also need to be optimized but in general the findings of this paper show that this methodology can be considered for further development in its application for labelling the full data set of Tsetse fly images.

# 4 References

Criminisi, A. and Shotton, J. eds., 2013. *Decision forests for computer vision and medical image analysis*. Springer Science & Business Media.

Cuingnet, R., Prevost, R., Lesage, D., Cohen, L.D., Mory, B. and Ardon, R., 2012, October. Automatic detection and segmentation of kidneys in 3D CT images using random forests. In *International Conference on Medical Image Computing and Computer-Assisted Intervention* (pp. 66-74). Springer, Berlin, Heidelberg.

Donner, R., Menze, B.H., Bischof, H. and Langs, G., 2013. Global localization of 3D anatomical structures by pre-filtered Hough Forests and discrete optimization. *Medical image analysis*, *17*(8), pp.1304-1314.

Hargrove, J., Hazelbag, M., 2019. Proposed project for a SACEMA masters thesis: An application of machine learning to landmark detection from two-dimensional images of wings of tsetse (*Glossina spp*)

Ibragimov, B., Likar, B. and Pernus, F., 2012. A game-theoretic framework for landmark-based image segmentation. *IEEE Transactions on Medical Imaging*, *31*(9), pp.1761-1776.

Josais, S., 2017.Automated identification of tsetse fly wing vein intersections.

Li, P., Prieto, L., Mery, D. and Flynn, P., 2018. Face Recognition in Low Quality Images: A Survey. *arXiv preprint arXiv:1805.11519*.

Lindner, C. and Cootes, T.F., 2015, May. Fully automatic cephalometric evaluation using random forest regression-voting. In *IEEE International Symposium on Biomedical Imaging*.

Louppe, G., 2014. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*.

Marée, R., Geurts, P. and Wehenkel, L., 2016. Towards generic image classification using tree-based learning: An extensive empirical study. *Pattern Recognition Letters*, *74*, pp.17-23.

Ouanan, H., Ouanan, M. and Aksasse, B., 2016, October. Facial landmark localization: Past, present and future. In *2016 4th IEEE International Colloquium on Information Science and Technology (CiSt)* (pp. 487-493). IEEE.

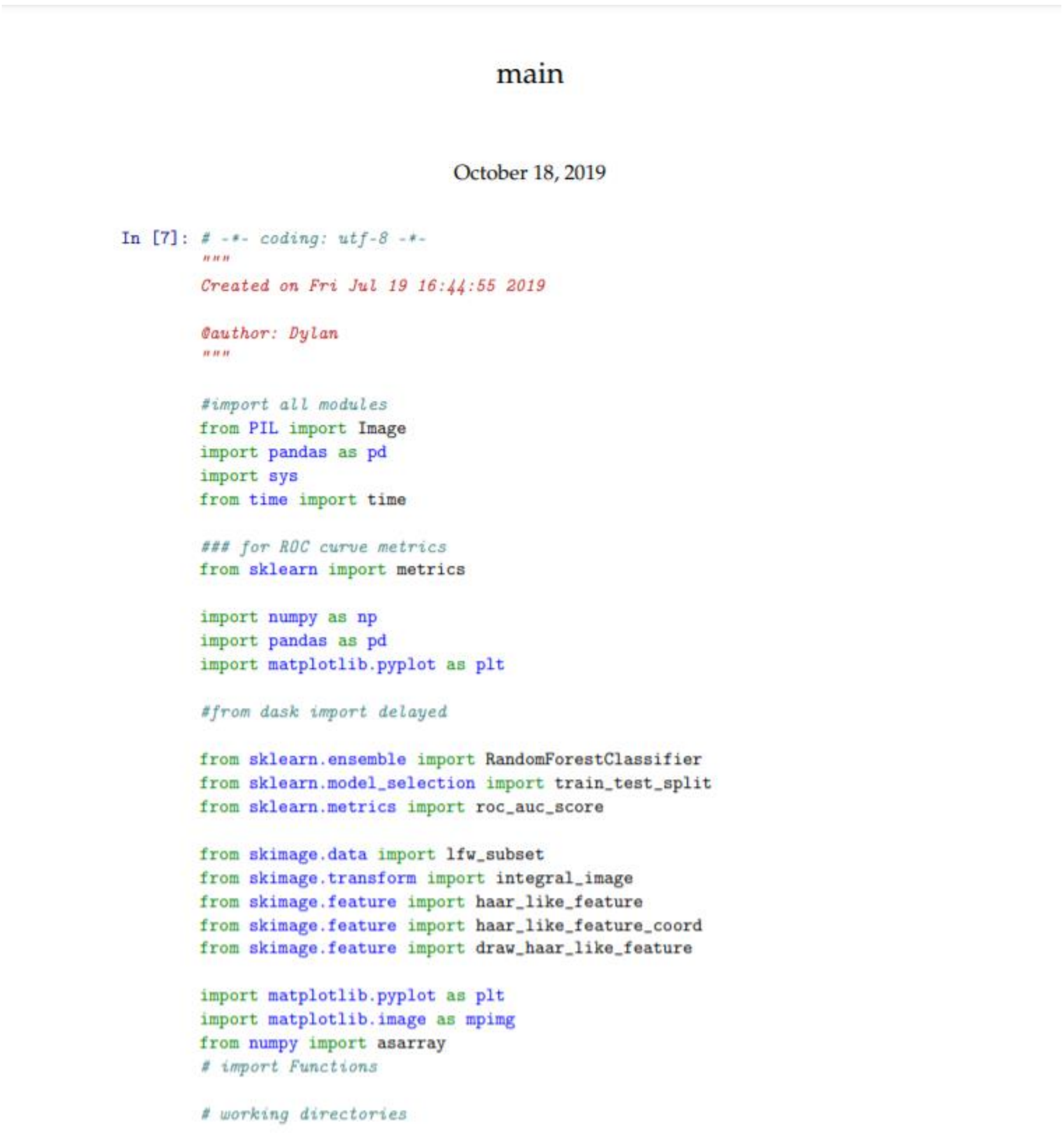
Vandaele, R., Aceto, J., Muller, M., Peronnet, F., Debat, V., Wang, C.W., Huang, C.T., Jodogne, S., Martinive, P., Geurts, P. and Marée, R., 2018. Landmark detection in 2D bioimages for geometric morphometrics: a multi-resolution tree-based approach. *Scientific reports*, *8*(1), p.538.

Viola, P. and Jones, M., 2001. Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, *1*(511-518), p.3.

Wade, B.S., Joshi, S.H., Pirnia, T., Leaver, A.M., Woods, R.P., Thompson, P.M., Espinoza, R. and Narr, K.L., 2015, April. Random forest classification of depression status based on subcortical brain morphometry following electroconvulsive therapy. In *2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI)* (pp. 92-96). IEEE.

Wang, C.W., Huang, C.T., Hsieh, M.C., Li, C.H., Chang, S.W., Li, W.C., Vandaele, R., Marée, R., Jodogne, S., Geurts, P. and Chen, C., 2015. Evaluation and comparison of anatomical landmark detection methods for cephalometric x-ray images: a grand challenge. *IEEE transactions on medical imaging*, *34*(9), pp.1890-1900.

Appendix I



# main

October 18, 2019

In [7]:
```
# -*- coding: utf-8 -*-
"""
Created on Fri Jul 19 16:44:55 2019

@author: Dylan
"""

#import all modules
from PIL import Image
import pandas as pd
import sys
from time import time

### for ROC curve metrics
from sklearn import metrics

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#from dask import delayed

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score

from skimage.data import lfw_subset
from skimage.transform import integral_image
from skimage.feature import haar_like_feature
from skimage.feature import haar_like_feature_coord
from skimage.feature import draw_haar_like_feature

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from numpy import asarray
# import Functions

# working directories
# --- change file path for you computer
# jeremy
# marijn
main = "/home/19007361/Dylan/" #"/home/dylan/Projects/Tetse_proj/Dylan/"
#data_2 = "/Home/CV_DYLAN/tetseP/Dylan/2/"
C_data2 =  "/home/19007361/Dylan/2/C_data2/"
Result_figs = "/home/19007361/Dylan/2/Result_figs/"
```

## 1 Create training data set

### 1.1 1) create training dataset of labeled coordinates for all sampled images

```
In [8]: # Read in labeled data from text file and create indexable dataframe
        """This code is to genereate a dataframe from the origional refined_data.txt,
            but we will use the processed csv file- new_textFile.csv"""

        #df = pd.read_fwf(C_data2 + 'refined_data.txt',header=None)
        #which = lambda lst:list(np.where(lst)[0])
        #new = df.iloc[:,4].str.split(" ", n = 23, expand = True)
        #df= df.drop(labels=4, axis=1)
        #text_data = pd.concat([df, new], axis=1, sort=False)
                # column numbers were messed up, make them go according to range
        #text_data.columns = range(text_data.shape[1])
        #text_data.reset_index(inplace=True, drop=True)

        #text_data = text_data.copy()

        text_data = pd.read_csv(main + 'new_textFile.csv')
        text_data.drop(text_data.columns[[0]], axis=1, inplace=True)

In [9]: # Create pixel sampling function for a single image

        def pix_sampling_training(image, imgname, i):

            coord_label_df = pd.DataFrame(columns=['label', 'coord', 'Image_name'])
            x1landmark = float(text_data.iloc[i,15])
            y1landmark = float(text_data.iloc[i,16])

            R= 9 ### with R 9 the second plot is quite hard to tell whether it works
            Rmax= 300

            npospix = 100
            nnegpix = 2*npospix

            a = np.random.uniform(0,1,npospix) * 2 * np.pi  #2 * pi * (0-1)

            a = np.random.uniform(0,1,npospix) * 2 * np.pi  #2 * pi * (0-1)
            r = R * np.sqrt(np.random.uniform(0,1,npospix)) # r *

            ## If you need it in Cartesian coordinates
```

```
    xwithin = r * np.cos(a)
    ywithin = r * np.sin(a)

    outer_radius =Rmax*Rmax
    inner_radius = R*R
    rho= np.sqrt(np.random.uniform(inner_radius,
                                 outer_radius, size=nnegpix))

    theta=  np.random.uniform( 0, 2*np.pi, nnegpix)
    xhoop = rho * np.cos(theta)
    yhoop = rho * np.sin(theta)

    for i in range(len(xwithin)):
        df2_within = pd.DataFrame({'label': ['pos'], 'coord': [(x1landmark +
                xwithin[i],y1landmark + ywithin[i])], 'Image_name':[imgname]})
        coord_label_df = coord_label_df.append(df2_within)
        #print(coord_label_df)
    for i in range(len(xhoop)):
        df2_hoop = pd.DataFrame({'label': ['neg'], 'coord': [(x1landmark +
                xhoop[i], y1landmark + yhoop[i])], 'Image_name': [imgname]})
        coord_label_df = coord_label_df.append(df2_hoop)
    return(coord_label_df)

def pix_sampling_prediction(image, imgname, i, std):

    coord_label_df = pd.DataFrame(columns=['label', 'coord', 'Image_name'])
    x1landmark = text_data.iloc[:,15]
    y1landmark = text_data.iloc[:,16]
    x1landmark_mean, y1landmark_mean = np.mean(x1landmark), np.mean(y1landmark)

    Rmax= std + 50

    npospix = 1000
    #nnegpix = 2*npospix

    a = np.random.uniform(0,1,npospix) * 2 * np.pi
    r = np.random.normal(0,Rmax,npospix)

    ## If you need it in Cartesian coordinates

    xwithin = r * np.cos(a)
    ywithin = r * np.sin(a)


    for i in range(len(xwithin)):
        df2_within = pd.DataFrame({'label': ['pos'], 'coord': [(x1landmark_mean +
                xwithin[i],y1landmark_mean + ywithin[i])], 'Image_name':[imgname]})
        coord_label_df = coord_label_df.append(df2_within)
        return(coord_label_df)

    # calculate mean and standard devation for the gaussian sampling
    #(used for predicition sampling only)

    #x1landmark = text_data.iloc[:,15]
    #y1landmark = text_data.iloc[:,16]
    #x1landmark_mean, y1landmark_mean = np.mean(x1landmark), np.mean(y1landmark)

    #distances = [np.sqrt((x1landmark_mean - text_data.iloc[i,15])**2
    #                                      + (y1landmark_mean -
    #                   text_data.iloc[i,16])**2) for i in range(len(x1landmark)) ]

    #std = np.std(distances)
```

```
In [11]: # Create a function that generates a dataset of labaled coordinates
         # using the pixel sampling function
         # NOTE: Change pix_sampling_training(..) to pix_sampling_prediction
         #to generate testing set testing set
         text_data = text_data.iloc[0:120,:]


         def coord_Tset(text_data):
             df = pd.DataFrame(columns=['label', 'coord', 'Image_name'])
             for i in range(text_data.shape[0]):
                 imgname=" ".join(text_data.iloc[i,[0,1,2]])
         #        file, ext = os.path.splitext(imgname)
                 img = mpimg.imread(C_data2 + imgname)
                 df = df.append(pix_sampling_training(img , imgname , i ))
             return(df.reset_index(drop=True))

         Tset = coord_Tset(text_data)
```

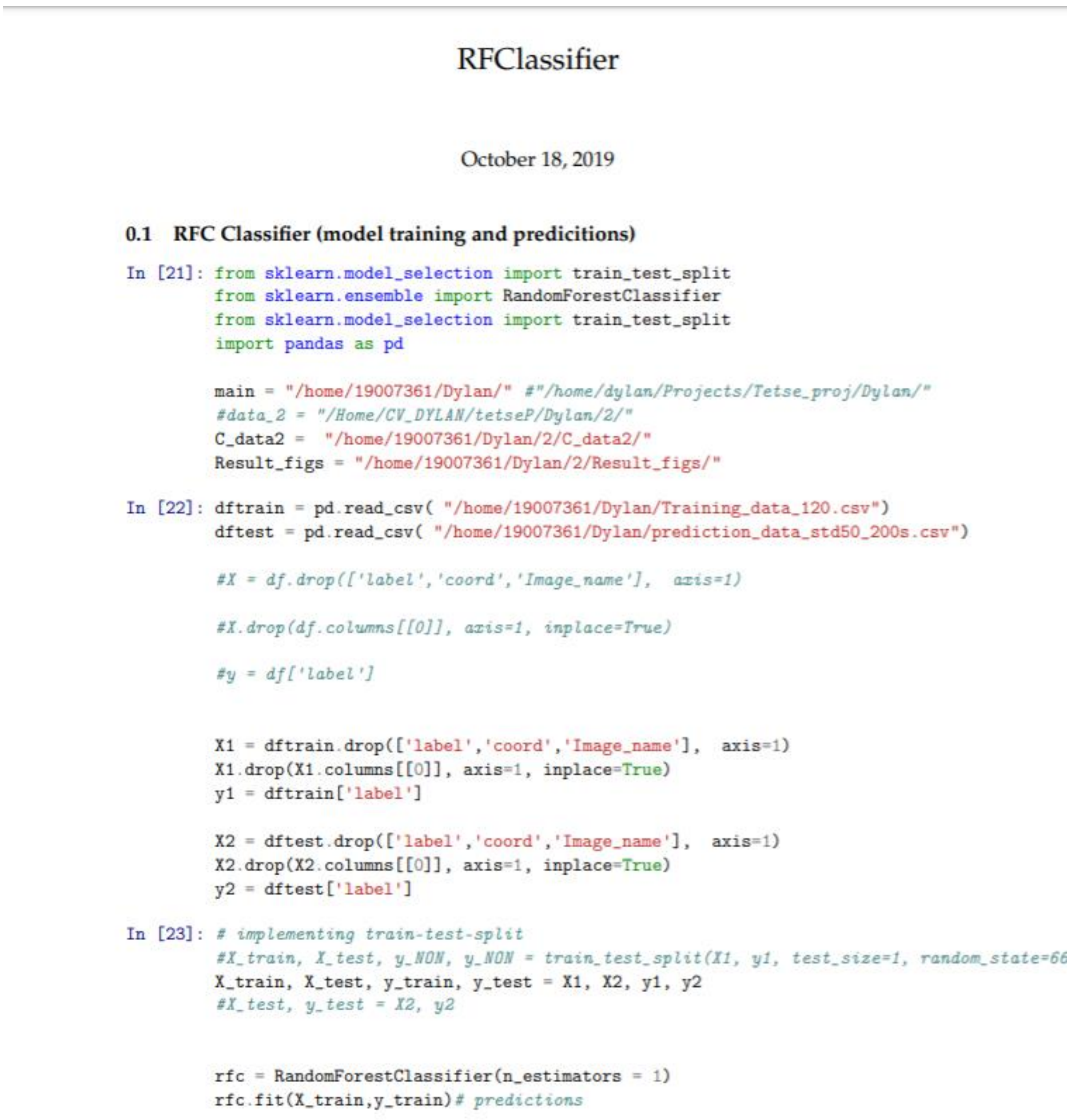
```
/home/19007361/.local/lib/python3.5/site-packages/pandas/core/frame.py:6692: FutureWarning: Sor
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  sort=sort)
```

## 2 2. Add Haarlike features for each pixel sample to the dataset

```
In [12]: #turn image to grey scale for inegral_image()
         #function in extract_feature_image()
```

```
        def rgb2gray(rgb):
            r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
            gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
            return gray

In [12]: def get_window(PIL_image, line , Crop=50, thumbnail=5):
            x,y = line.iloc[0,1]
            img_cropped = PIL_image.crop((x- Crop,y-Crop,x+Crop,y+Crop) )
            plt.show(img_cropped)
            plt.show()
            img_cropped.thumbnail([thumbnail,thumbnail])
            return img_cropped


        feature_types = ['type-4' , 'type-2-x', 'type-2-y','type-3-x', 'type-3-y']

        def extract_feature_image(img, feature_types, feature_coord=None):
            """Extract the haar feature for the current image"""
            ii = integral_image(img)
            features = []
            for i in range(len(feature_types)):
                features.extend(haar_like_feature(ii, 0, 0, ii.shape[0], ii.shape[1],
                                     feature_types[i],
                                     feature_coord=None))
            return features

In [14]: # The Function below extracts haarlike features from
        #  a window around each pixel sample
        # and appends it to the the dataframe.

        # INPUT: The coordinate dataset generated from coord_Tset



        def features_Tset(Tset):
            df_final = pd.DataFrame([])
            for i in range(len(Tset)):
                line = Tset.iloc[[i]]
                line = line.reset_index(drop=True)
                imgname = line.iloc[0,0]

                img = Image.open(C_data2 + imgname)

                window = get_window(img, line)

                features = extract_feature_image(rgb2gray(asarray(window))
                                                 , feature_types)
                        s = pd.DataFrame([features])

                        data = line.join(s)

                        df_final = df_final.append(data)
                        df_final = df_final.reset_index(drop = True)
                    return  df_final.to_csv(main + 'Training_data_120.csv')

            #features_Tset(Tset)
```

## Appendix II



# RFClassifier

October 18, 2019

### 0.1 RFC Classifier (model training and predicitions)

```
In [21]: from sklearn.model_selection import train_test_split
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.model_selection import train_test_split
         import pandas as pd

         main = "/home/19007361/Dylan/" #"/home/dylan/Projects/Tetse_proj/Dylan/"
         #data_2 = "/Home/CV_DYLAN/tetseP/Dylan/2/"
         C_data2 =  "/home/19007361/Dylan/2/C_data2/"
         Result_figs = "/home/19007361/Dylan/2/Result_figs/"

In [22]: dftrain = pd.read_csv( "/home/19007361/Dylan/Training_data_120.csv")
         dftest = pd.read_csv( "/home/19007361/Dylan/prediction_data_std50_200s.csv")

         #X = df.drop(['label','coord','Image_name'],  axis=1)

         #X.drop(df.columns[[0]], axis=1, inplace=True)

         #y = df['label']


         X1 = dftrain.drop(['label','coord','Image_name'],  axis=1)
         X1.drop(X1.columns[[0]], axis=1, inplace=True)
         y1 = dftrain['label']

         X2 = dftest.drop(['label','coord','Image_name'],  axis=1)
         X2.drop(X2.columns[[0]], axis=1, inplace=True)
         y2 = dftest['label']

In [23]: # implementing train-test-split
         #X_train, X_test, y_NON, y_NON = train_test_split(X1, y1, test_size=1, random_state=66
         X_train, X_test, y_train, y_test = X1, X2, y1, y2
         #X_test, y_test = X2, y2


         rfc = RandomForestClassifier(n_estimators = 1)
         rfc.fit(X_train,y_train)# predictions
```

```
rfc_predict = rfc.predict(X_test)
#print(X_test.iloc[[0]].index.values.astype(int))
```

## 1 NEXT : plot the predictions on the images

```
In [24]: #First we find the row index of the the pos predictions
         pos_index = []
         for i in range(len(rfc_predict)):
             if rfc_predict[i] == 'pos':

                 index = X_test.iloc[[i]].index.values.astype(int)[0]

                 pos_index.append(index)

         coordinates =  []
         index = pos_index[0]

         index = int(index)

         # Then create A dataframe containing all the coordinates with image name of the positi
         #predictions found using the index
         for j in range(len(pos_index)):
             index = int(pos_index[j])
             line = dftest.iloc[[index]]
             coord = [eval(line.iloc[0,2]),line.iloc[0,1]]
             coordinates.append(coord)

         cdf = pd.DataFrame(coordinates.copy())

In [25]: from PIL import Image
         import matplotlib.pyplot as plt
         import statistics as stat

         image_names = {}
         for cat in coordinates:
             if not cat[1] in image_names:
                 image_names[cat[1]] = [cat[0]]
                 #print(dylan[cat[1]])
             else:
                 image_names[cat[1]].append(cat[0])

         image1_name = list(image_names)[0]
         #print(image1_name)
         im = Image.open(C_data2 + image1_name)
         plt.imshow(im)

         for dog in image_names[image1_name]:
```

```
    #print(dog[0])
    plt.plot(dog[0],dog[1],'o')
```

```
In [26]: import math
         import numpy as np
         text_data = pd.read_csv(main + 'new_textFile.csv')
         text_data.drop(text_data.columns[[0]], axis=1, inplace=True)
         text_data

         #def reject_outliers(data, m=2.):
         #    return data[abs(data - np.mean(data)) < m * np.std(data)]

         def reject_outliers(data, m = 2., col = 2):
             d = np.abs(data.iloc[:,4] - np.median(data.iloc[:,4]))
             mdev = np.median(d)
             s = d/mdev if mdev else 0.
             return data[s<m]

         final_landmarks = []
         for name in list(image_names):
             df_coord = pd.DataFrame(image_names[name])
             x = list(df_coord.iloc[:,0])
             y = list(df_coord.iloc[:,1])
             x_median = stat.median(x)
             y_median = stat.median(y)
             for i in range(text_data.shape[0]):
                 imgname=" ".join(text_data.iloc[i,[0,1,2]])
                 if imgname == name:

                     dist = math.sqrt((abs(text_data.iloc[i,15] - x_median))**2 + (abs(text_data
                                                                                     y_median)

                     final_landmarks.append([text_data.iloc[i,15],text_data.iloc[i,16],x_median,



         f = pd.DataFrame(final_landmarks)
```

```
In [28]: fr = reject_outliers(f,m = 1.0)
```

```
In [29]: df = f.copy()

         distances = list(df.iloc[:,4])

         x = range(len(distances))
         mean = stat.mean(distances)
         plt.bar(x,distances,color = 'b')
         plt.axhline(y=mean, color='r', linestyle='-')
```

```
        plt.title('The distance error from true landmark (with outliers)')
        plt.xlabel('Images')
        plt.ylabel('Pixel distance error')
        plt.savefig(Result_figs + 'DistanceERROR(WITH_d1).png')
        #plt.show()

In [30]: import numpy as np
        # Calculate the average
        mean = np.mean(distances)

        # Calculate the standard deviation
        std = np.std(distances)

        # Define labels, positions, bar heights and error bar heights
        labels = ['landmark no.7']
        x_pos = np.arange(len(labels))
        CTEs = [mean]
        error = [std]


        # Build the plot
        fig, ax = plt.subplots()
        ax.bar(x_pos, CTEs,
               yerr=error,
               align='center',
               alpha=0.5,
               ecolor='black',
               capsize=10)
        ax.set_ylabel('pixel distance')
        ax.set_xticks(x_pos)
        ax.set_xticklabels(labels)
        ax.set_title('Average distance from true landmark (with outliers)')
        ax.yaxis.grid(True)


        # Save the figure and show
        plt.tight_layout()
        plt.savefig(Result_figs + 'stdERROR(WITH).png')
        #plt.savefig('bar_plot_with_error_bars.png')
        #plt.show()

In [31]: df = fr.copy()

        distances = list(df.iloc[:,4])

        x = range(len(distances))
        mean = stat.mean(distances)
        plt.bar(x,distances,color = 'b')
        plt.axhline(y=mean, color='r', linestyle='-')
        plt.title('The distance error from true landmark (without outliers)')
        plt.xlabel('Images')
        plt.ylabel('Pixel distance error')
        plt.savefig(Result_figs + 'DistanceERROR(WITHout_d1).png')
        #plt.show()

In [32]: import numpy as np
        # Calculate the average
        mean = np.mean(distances)

        # Calculate the standard deviation
        std = np.std(distances)

        # Define labels, positions, bar heights and error bar heights
        labels = ['landmark no.7']
        x_pos = np.arange(len(labels))
        CTEs = [mean]
        error = [std]


        # Build the plot
        fig, ax = plt.subplots()
        ax.bar(x_pos, CTEs,
               yerr=error,
               align='center',
               alpha=0.5,
               ecolor='black',
               capsize=10)
        ax.set_ylabel('pixel distance')
        ax.set_xticks(x_pos)
        ax.set_xticklabels(labels)
        ax.set_title('Average distance from true landmark (without outliers)')
        ax.yaxis.grid(True)


        # Save the figure and show
        plt.tight_layout()
        plt.savefig(Result_figs + 'stdERROR(WITHout_d1).png')
        #plt.show()
        #print('Standard deviation (without outliers)= ' , error)
        #print('mean = ' , CTEs)

In [35]: # plot all the posititve landmarks on the coresponding images

        from PIL import Image
        import numpy as np
        image_names = {}
        for cat in coordinates:
            if not cat[1] in image_names:
                image_names[cat[1]] = [cat[0]]

            else:
                image_names[cat[1]].append(cat[0])
```

```
In [18]: #This code plots the predictions on there image

        for i in range(len(list(image_names))):
            image1_name = list(image_names)[i]

            im = C_data2 + image1_name
            immy = plt.imread(im)
            x = list(image_names[image1_name])[0][0]
            y = list(image_names[image1_name])[0][1]
            x1 = (int(x-300))
            y1 = (int(y-300))
            x2 = (int(x+300))
            y2 = (int(y+300))
            im_crop = immy[y1:y2, x1:x2, : ]

            plt.imshow(im_crop)

            for dog in image_names[image1_name]:
                xd = x - 300
                yd = y - 300
                plt.plot(dog[0] - xd,dog[1] - yd,'o', ms = 2)
                x = dog[0]
                y=dog[1]
            plt.savefig(Result_figs + image1_name)
        #plt.show()
```