Automatic detection of honeybees in a hive

Mihai Iulian Florea
Abstract

Automatic detection of honeybees in a hive

Mihai Iulian Florea

The complex social structure of the honey bee hive has been the subject of inquiry since the dawn of science. Studying bee interaction patterns could not only advance sociology but find applications in epidemiology as well. Data on bee society remains scarce to this day as no study has managed to comprehensively catalogue all interactions among bees within a single hive. This work aims at developing methodologies for fully automatic tracking of bees and their interactions in infrared video footage.

H.264 video encoding was investigated as a means of reducing digital video storage requirements. It has been shown that two orders of magnitude compression ratios are attainable while preserving almost all information relevant to tracking.

The video images contained bees with custom tags mounted on their thoraxes walking on a hive frame. The hive cells have strong features that impede bee detection. Various means of background removal were studied, with the median over one hour found to be the most effective for both bee limb and tag detection. K-means clustering of local textures shows promise as an edge filtering stage for limb detection.

Several tag detection systems were tested: a Laplacian of Gaussian local maxima based system, the same improved with either support vector machines or multilayer perceptrons, and the Viola-Jones object detection framework. In particular, this work includes a comprehensive description of the Viola-Jones boosted cascade with a level of detail not currently found in literature. The Viola-Jones system proved to outperform all others in terms of accuracy. All systems have been found to run in real-time on year 2013 consumer grade computing hardware. A two orders of magnitude file size reduction was not found to noticeably reduce the accuracy of any tested system.
# Contents

Abbreviations vii

1 Introduction 1

2 Materials 3
   2.1 Bee Hive 3
      2.1.1 Tags 3
   2.2 Video 4
      2.2.1 Video Files 4
   2.3 Frame Dataset 7
   2.4 Computing environment 9

3 Video Preprocessing 10
   3.1 Video compression 10
      3.1.1 Cropping 10
      3.1.2 Huffman YUV 11
      3.1.3 H.264/AVC 11
      3.1.4 x264 performance 15
   3.2 Background removal 23
      3.2.1 Clustering 25
      3.2.2 Frame differencing 32
      3.2.3 Exponentially weighted moving average 32
      3.2.4 True average 36
      3.2.5 Histogram peak 36
      3.2.6 Percentile 37
      3.2.7 Mixture of Gaussians 39
   3.3 Discussion and future work 43

4 Tag detection 45
   4.1 Why is tag detection difficult? 45
   4.2 LoG pipeline 47
   4.3 Viola-Jones object detection framework 52
      4.3.1 Haar-like Features 52
      4.3.2 Adaboost 56
      4.3.3 Boosted cascade 63
      4.3.4 Results 71
   4.4 Benchmarks 74
      4.4.1 Multilayer perceptrons 74
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFPT</td>
<td>Average Frame Processing Time</td>
</tr>
<tr>
<td>ARTag</td>
<td>Augmented Reality Tag</td>
</tr>
<tr>
<td>AVC</td>
<td>Advanced Video Coding</td>
</tr>
<tr>
<td>B-frame</td>
<td>Bidirectional frame</td>
</tr>
<tr>
<td>BMB</td>
<td>B-frame Macroblock</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DCT</td>
<td>Discrete Cosine Transform</td>
</tr>
<tr>
<td>DDR</td>
<td>Double Data Rate</td>
</tr>
<tr>
<td>DPCM</td>
<td>Differential Pulse Code Modulation</td>
</tr>
<tr>
<td>DVQ</td>
<td>Digital Video Quality</td>
</tr>
<tr>
<td>EWMA</td>
<td>Exponentially Weighted Moving Average</td>
</tr>
<tr>
<td>FFMPEG</td>
<td>Fast Forward Moving Picture Experts Group</td>
</tr>
<tr>
<td>fn</td>
<td>false negative count</td>
</tr>
<tr>
<td>fp</td>
<td>false positive count</td>
</tr>
<tr>
<td>FPR</td>
<td>False Positive Rate</td>
</tr>
<tr>
<td>fps</td>
<td>frames per second</td>
</tr>
<tr>
<td>GCC</td>
<td>GNU Compiler Collection</td>
</tr>
<tr>
<td>Gib</td>
<td>Gibibyte (1073741824 bytes)</td>
</tr>
<tr>
<td>GNU</td>
<td>GNU’s Not Unix (recursive acronym)</td>
</tr>
<tr>
<td>GPL</td>
<td>General Public License</td>
</tr>
<tr>
<td>HDD</td>
<td>Hard Disk Drive</td>
</tr>
<tr>
<td>I-frame</td>
<td>Intra-coded frame</td>
</tr>
<tr>
<td>iDCT</td>
<td>inverse Discrete Cosine Transform</td>
</tr>
<tr>
<td>JDD</td>
<td>Just Noticeable Difference</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Term</td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
</tr>
<tr>
<td>kB</td>
<td>kilobyte (1000 bytes)</td>
</tr>
<tr>
<td>KiB</td>
<td>Kibibyte (1024 bytes)</td>
</tr>
<tr>
<td>LED</td>
<td>Light-Emitting Diode</td>
</tr>
<tr>
<td>LoG</td>
<td>Laplacian of Gaussian</td>
</tr>
<tr>
<td>MB</td>
<td>Macroblock</td>
</tr>
<tr>
<td>MB</td>
<td>Megabyte (1000000 bytes)</td>
</tr>
<tr>
<td>MLC</td>
<td>Multi-Level Cell</td>
</tr>
<tr>
<td>MLP</td>
<td>Multilayer Perceptrons</td>
</tr>
<tr>
<td>MNIST</td>
<td>Mixed National Institute of Standards and Technology dataset</td>
</tr>
<tr>
<td>MOG</td>
<td>Mixture of Gaussians</td>
</tr>
<tr>
<td>MPEG</td>
<td>Moving Picture Experts Group</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Squared Error</td>
</tr>
<tr>
<td>OpenCV</td>
<td>Open Source Computer Vision Library</td>
</tr>
<tr>
<td>P-frame</td>
<td>Predicted frame</td>
</tr>
<tr>
<td>PMB</td>
<td>P-frame Macroblock</td>
</tr>
<tr>
<td>QP</td>
<td>Quantization Parameter</td>
</tr>
<tr>
<td>RAM</td>
<td>Random-Access Memory</td>
</tr>
<tr>
<td>RANSAC</td>
<td>Random Sample Consensus</td>
</tr>
<tr>
<td>RBFNN</td>
<td>Radial Basis Function Neural Network</td>
</tr>
<tr>
<td>RGB</td>
<td>Red Green Blue</td>
</tr>
<tr>
<td>ROC curve</td>
<td>Receiver Operating Characteristic curve</td>
</tr>
<tr>
<td>RPM</td>
<td>Revolutions Per Minute</td>
</tr>
<tr>
<td>RProp</td>
<td>Resilient Propagation</td>
</tr>
<tr>
<td>SATA</td>
<td>Serial Advanced Technology Attachment</td>
</tr>
<tr>
<td>SSD</td>
<td>Solid-State Drive</td>
</tr>
<tr>
<td>SSIM</td>
<td>Structural Similarity</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>tn</td>
<td>true negative count</td>
</tr>
<tr>
<td>tp</td>
<td>true positive count</td>
</tr>
<tr>
<td>TPR</td>
<td>True Positive Rate</td>
</tr>
</tbody>
</table>
VCR  Video Cassette Recording
YCrCb  Luma (Y), Chrominance red (Cr) and Chrominance blue (Cb) color space
YUV  Color space made up of a luma (Y) and two chrominance components (UV)
Chapter 1

Introduction

Honey bees \((\textit{Apis mellifera})\) exhibit many forms of intelligent behavior being the only species, apart from humans, that are able to communicate directions \[1\]. Given their complex social structure, where individuals have clearly defined roles, it is very likely that interactions among bees could bear resemblance with those among humans.

Human sociological studies are limited in their effectiveness due to restrictions in data collection methods. A bee colony on the other hand is self-contained, with few social interactions outside it. Hives can be artificially modified by experimenters who may open them completely in order to observe every interaction. This can enable cataloging all honeybee motions and provide valuable data to social sciences. Information on disease transmission in social groups is of particular interest \[2\].

Scientific inquiry into honey bee behavior stems back to antiquity \[3\]. Aristotle mentions the bee waggle dance, uncertain of its meaning. He also observed similarities between human and honey bee societies, grouping both species into the category of "social animals".

Von Frisch \[1\] has proven that honey bees are capable of communicating directions through the waggle dance. Experimentation and data collection had to be carried out manually, which limited the accuracy and quantity of information obtained.

More recently, computer assisted tracking of bee movements has been accomplished \[4\]. Through these studies, trajectories of single bees have been automatically mapped from video recordings using Probabilistic Principal Component Analysis for intra-frame position recognition and Rao-Blackwellized Particle Filters for inter-frame trajectory prediction. Excellent results were obtained without the aid of any markers on the bees. However, tracking a single individual offers little insight into communication and disease transmission.

Hundreds of bees at a time have been tracked in video sequences \[5\] with the aid of large circular markers painted on their thoraxes. Unfortunately, the trajectories extracted do not contain head orientation data that are necessary for the detection of trophallaxis - the transfer of food between bees by means of their tongues \[6\]. Separating the camera and hive using a transparent screen allowed bees to walk on its surface, occluding the marker.

By marking both the dorsal and ventral parts of the abdomen with a large marker, more consistent data has been obtained \[7\]. Again the head orientation
A very ambitious study [8] has managed to devise a method for extracting the posture of hundreds of bees at a time from very low resolution video. By approximating the shape of honey bee bodies by an oval of constant size throughout the sequence, head posture information has been inferred with a reasonable degree of accuracy. The video images have been segmented using Vector Quantization (a form of clustering) and a separate post-processing step has been employed to separate touching bees. Analysis was limited to a few minutes of video. In addition, the hive was illuminated with red light that may have altered bee behavior [6].

The current work, initiated at the Department of Ecology of the Swedish University of Agricultural Sciences (SLU), caters to the need of developing better methodologies for fully automated tracking of all movements of all the bees in a single hive, including head and antennae positions.

Given the complexity of the tasks at hand, the scope of this work will be limited to achieving the following objectives:

1. Find a methodology to reduce as much as possible the size of the recorded video while preserving relevant details. Storing hive videos totaling several weeks in length at a resolution high enough to allow the identification of individual bees is beyond the capability of 2013 consumer storage technology. At least an order of magnitude size reduction is necessary to make long term recording feasible at this point.

2. Determine whether it is feasible to track all the bees in real-time. Should this be possible, only interaction data would need to be recorded, greatly reducing the storage requirements.

3. All software platforms utilized in this work ought to be made entirely of free [9] or at least open source software. The availability of the code provides several advantages. First and foremost, it adheres to the academic principle of openness. Second, it makes the methodology reproducible. And lastly, compiling source code instead of using prebuilt binaries leads to increased performance, necessary when dealing with large amounts of data.
Chapter 2

Materials

Researchers at the SLU and Uppsala University have recorded raw video footage of bees for offline analysis with the hope that the methods developed on recorded video can be sped up sufficiently to allow real time analysis [2].

2.1 Bee Hive

The honey bees were filmed in a standard observation hive (width 43.5 cm, height 52.5 cm, depth 5.5 cm) containing two standard hive frames (width 37 cm and height 22 cm each) mounted vertically one on top of the other. Two Plexiglas sheets found on both sides of the hive were used to contain the bees. The hive was placed in a small, dark, windowless room and was sealed so that no bee could enter or leave the hive during filming. Bees were kept alive by dripping sugar water into the hive. To simplify the experiment, bees were marked with tags placed only on their backs. In order to prevent the bees from walking on the screen and thus occluding their tags, the experimenters sprayed the screen with a thin film of Fluon, a slippery coating agent, with the help of an air-brush instrument.

2.1.1 Tags

Generally, bee-keepers are interested in tagging only the queen of each hive. The tags they use are small, circular (of 3 mm in diameter) and inscribed with Arabic numerals. This method however cannot be extrapolated to the high number of bees simultaneously tracked in this experiment. Consequently, an innovative square tag design was chosen instead (fig. 2.1) [2]. The tags are square in shape, of size 3 mm by 3 mm. The bright white rectangle (gray level 255 on a 0 to 255 scale) in the center is used in the detection of the bee. The tag is glued on the dorsal part of the thorax of the bee with the white line emerging from the center pointing towards the head. The 8 rectangular patches, marked \(c_0, c_1, ..., c_7\) are homogeneous, with the gray level encoding a base 3 digit: (0 to encode digit 0, 65 for 1, 130 for 2). The number encoded by the tag is given by \(c_0 \cdot 3^7 + c_1 \cdot 3^6 + ... + c_7 \cdot 3^0\) yielding an ID range of 0 to 6560.
2.2 Video

A Basler Scout scA1600-14gm camera mounted with Fujinon HF16HA-1B lens and 850 nm bandpass filter was used to film the hive. LED lights at 850 nm were used for illumination. Instead of employing a diffusion system, the frame was lit from a wide angle with respect to the camera, so that no specular reflections from the screen would enter the field of view. Bees are thought to be insensitive to near infrared light [1] and should behave as if in total darkness.

The distance between the camera and the hive frame was of 80 cm so that the field of view encompassed the entire frame, with a small margin. The optical axis is perpendicular to the center of the frame. A typical video frame is shown in fig. 2.2.

Filming took place over the course of five days. Due to limitations in hard-disk capacity, only around 11 hours of video were recorded at a time with breaks in between for computer servicing and cleaning of the glass.

The video was recorded using frames of size $1626 \times 1236$ pixels at a frequency of 14.3 fps. The video frames contain only one 8 bit channel corresponding to 850 nm near-infrared light intensity. The encoding format is lossless Huffman YUV compression [10].

2.2.1 Video Files

The entire video material comprises 10 files totaling around 5 continuous days of footage. The video files with their corresponding lengths are listed in Table 2.1. A detailed description of their contents is as follows:

d1_140812.avi The image is very sharp and the tags are clearly visible. Around 150 live bees are present. The hive has a queen, which is surrounded by bees tending it. Sugar water drips from the top of the hive to keep the bees alive. A large number of bees lie dead at the bottom of the frame and there are a few bodies higher up. Some bees have ripped the tags from their backs and these tags, whole or in pieces, can be found in various points across the frame. After 1 hour, the liquid produces splash marks in the lower part of the screen. Bees are clumped around the extremities of the hive initially and form two clumps in the upper part of the frame towards the end of the video.
Figure 2.2: A typical video frame. The size in pixels is $1626 \times 1236$. The image is in grayscale format with one 8-bit channel. The original film was upside down. Here the frame is shown after being rotated $180^\circ$.

d1_150812.avi The same hive as in the previous video is filmed. The queen is still present and the bees gather around it. For this reason, almost all the bees are located in the left side of the frame while little activity can be seen on the right side. The liquid splashes are visible from the very beginning and remain a problem throughout the video.

d1_150812_extra.avi Almost the same as the previous video with the exception that the right side of the frame has a few active bees.

d2_210812.avi A different beehive is filmed from now on. No queen is present and the hive is better lit than in the previous videos. The image is not very sharp although the hive does not have debris nor dripping liquid. In this sequence, the bees are very inactive and form a single clump that moves slowly around the frame. In the end, the video looks more blurry, most likely because the breath of the bees fogs up the screen.

d2_210812_del2.avi The video starts with all bees forming a single large clump. During the next 6 hours, the clump moves around slightly and then splits into two less dense clumps starting from the 8th hour. After the 9th hour, the bees are spread out somewhat with occasional crowding. During the first 12 hours, the bees are very stationary. Starting with the 13th hour the bees start moving around.

d3_220812_del1_LON.avi For the first 3 hours, the bees move around en-
Table 2.1: Currently Available Video Files

<table>
<thead>
<tr>
<th>Video File Name</th>
<th>Duration</th>
<th>Hard-Disk ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1_140812.avi</td>
<td>11:15:39.91</td>
<td>HDD 01</td>
</tr>
<tr>
<td>d1_150812.avi</td>
<td>10:25:37.38</td>
<td>HDD 02</td>
</tr>
<tr>
<td>d1_150812_extra.avi</td>
<td>00:21:33.77</td>
<td>HDD 02</td>
</tr>
<tr>
<td>d2_210812.avi</td>
<td>05:34:48.37</td>
<td>HDD 03</td>
</tr>
<tr>
<td>d2_210812_del2.avi</td>
<td>12:12:52.75</td>
<td>HDD 03</td>
</tr>
<tr>
<td>d3_220812_del1_LON.avi</td>
<td>03:55:36.98</td>
<td>HDD 04</td>
</tr>
<tr>
<td>d3_220812_del2_LOFF.avi</td>
<td>05:41:27.60</td>
<td>HDD 04</td>
</tr>
<tr>
<td>d3_230812.avi</td>
<td>17:20:04.97</td>
<td>HDD 05</td>
</tr>
<tr>
<td>d4_240812.avi</td>
<td>23:20:47.19</td>
<td>HDD 06</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>90:08:28.92</td>
<td></td>
</tr>
</tbody>
</table>

- Table entry: energetically. Because the bees do not clump together during this period, tags are clearly visible. Later on, bee activity decreases. Relatively few dead bees can be found at the bottom of the frame.

  **d3_220812_del2_LOFF.avi** In the beginning, bees do not move as much as in the previous sequence though bees are well spread out and quite active. During the last 2 hours, almost all bees move quickly around the frame. The number of dead bees remains low.

  **d3_230812.avi** The frame is well lit. Bees initially linger in the center. During the 4th hour, the bees become more active and start moving around. A few dozen bees escape the enclosure. In the 8th hour bees clump very tightly into a single cluster. The clump remains until the 16th hour after which the bees disperse and move freely. The glass is very foggy in the end.

  **d4_240812.avi** From the 7th hour onward, most bees leave the hive. The remaining are either grouped in small clumps or lie dead at the bottom of the hive frame. By the 23rd hour, a single small clump remains in the top left corner. At this point, most of the hive frame lies exposed and unaltered from the beginning.

  The last hour could be used to infer the background to be used in the preprocessing the first 6 hours.

The following sections will focus on the analysis to the first hour of the file d1_140812.avi. It is one of the 3 files that depict a realistic bee colony complete with its own queen. Bees move slowly, which allows for more accurate tracking given the fixed 14.3 fps framerate. The bee bodies and dripping liquid model a more realistic scenario where bees are confined for longer spans of time. The first hour does not show any liquid splashes, which are difficult to account for when looking for tags.
2.3 Frame Dataset

In order to conduct machine learning experiments, 6 frames in the video were manually marked. A filled red circle of radius 4 pixels was drawn on top of every visible tag. The circle center was chosen represent as accurately as possible the center of the tag (figure 2.3). Although the dataset materials are made up of images, only the marked tag center coordinates are used. The image data around the tag centers is extracted either from either the raw video frames, or the corresponding compressed video frames. Hence, for every compression settings, a distinct image dataset is created based on the coordinate dataset.

![Figure 2.3: Typical dataset frame. Tags are marked with red dots.](image)

3 frames were used as a training set. 3 separate test sets were created, each from a single frame. The test set frames were chosen for their properties with respect to compression. The first has the least information discarded and represents a best case scenario for a given compression setting. The second represents the average case while the third the worst case. All frames were chosen more than 1 minute apart in order to avoid duplicate tag positions. Furthermore, the test frames occur 10 minutes later than the training sets to penalize systems that simply memorize training data. Frames that are farther apart in time have less data in common.

The frame indices, display times as well as the effect that compression has on each frame are listed in table 2.2. The x264 encoder [11] with global scalar quantizer setting [12] of QP = 4 produces high squared error variation among the selected frames. The squared error is an indicator of how quality degrades with higher compression settings.

The coordinate datasets based on the above mentioned frames are listed in table 2.3.
### Table 2.2: List of marked frames

<table>
<thead>
<tr>
<th>Frame index</th>
<th>Frame label</th>
<th>Compressed frame type</th>
<th>Square Error (QP = 4)</th>
<th>Display time</th>
<th>Marked tag count</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>A</td>
<td>I frame</td>
<td>39691</td>
<td>00:11:39.37</td>
<td>160</td>
</tr>
<tr>
<td>11248</td>
<td>B</td>
<td>B frame</td>
<td>337693</td>
<td>00:13:06.57</td>
<td>147</td>
</tr>
<tr>
<td>12375</td>
<td>C</td>
<td>P frame</td>
<td>149246</td>
<td>00:14:25.38</td>
<td>127</td>
</tr>
<tr>
<td>20001</td>
<td>D</td>
<td>I frame</td>
<td>39132</td>
<td>00:23:18.67</td>
<td>147</td>
</tr>
<tr>
<td>21248</td>
<td>E</td>
<td>B frame</td>
<td>337050</td>
<td>00:24:45.87</td>
<td>137</td>
</tr>
<tr>
<td>22375</td>
<td>F</td>
<td>P frame</td>
<td>148437</td>
<td>00:26:04.68</td>
<td>147</td>
</tr>
</tbody>
</table>

### Table 2.3: List of datasets

<table>
<thead>
<tr>
<th>Dataset name</th>
<th>Description</th>
<th>Frames</th>
<th>Marked tag count</th>
</tr>
</thead>
<tbody>
<tr>
<td>train-3</td>
<td>Standard training set</td>
<td>A, B, C</td>
<td>434</td>
</tr>
<tr>
<td>test-hq</td>
<td>Highest Quality Test</td>
<td>D</td>
<td>147</td>
</tr>
<tr>
<td>test-lq</td>
<td>Lowest Quality Test</td>
<td>E</td>
<td>137</td>
</tr>
<tr>
<td>test-mq</td>
<td>Average Quality Test</td>
<td>F</td>
<td>147</td>
</tr>
</tbody>
</table>
2.4 Computing environment

The computing hardware used in all experiments consisted of a consumer grade desktop computer with all components produced in the year 2013. Staying true to the objectives of this project, all software utilized in this work was released under an open-source license. Most was part of the GNU/Linux system released under the GPL license. Some video compression components are under non-free licenses but allow royalty free usage for academic purposes. The full list of the hardware and software components can be found in table 2.4.

<table>
<thead>
<tr>
<th>Hardware Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz</td>
</tr>
<tr>
<td></td>
<td>8 virtual cores corresponding to 4 physical cores</td>
</tr>
<tr>
<td>RAM</td>
<td>8050176 kB DDR3</td>
</tr>
<tr>
<td>Main disk</td>
<td>Intel 330 Series 2.5” 120 GB SSD SATA/600 MLC 25 nm</td>
</tr>
<tr>
<td></td>
<td>(INTEL SSDSC2CT120A3)</td>
</tr>
<tr>
<td>Video file disk</td>
<td>WD BLACK 3.5” 3TB 7200 RPM SATA/600</td>
</tr>
<tr>
<td></td>
<td>(WDC WD5000AAKX-60U6AA0) Buffer Size: 16384kB</td>
</tr>
<tr>
<td>Kernel</td>
<td>Linux Kernel 3.9.4-200 x86_64</td>
</tr>
<tr>
<td>Compiler</td>
<td>gcc (GCC) 4.7.2 20121109</td>
</tr>
<tr>
<td>Video transcoding software</td>
<td>FFMpeg version N-52339-g0dd25e4</td>
</tr>
<tr>
<td></td>
<td>configuration: --enable-gpl --enable-libx264 --enable-nonfree</td>
</tr>
<tr>
<td></td>
<td>libavutil 52. 27.100 / 52. 27.100</td>
</tr>
<tr>
<td></td>
<td>libavcodec 55. 5.100 / 55. 5.100</td>
</tr>
<tr>
<td></td>
<td>libavformat 55. 3.100 / 55. 3.100</td>
</tr>
<tr>
<td></td>
<td>libavdevice 55. 0.100 / 55. 0.100</td>
</tr>
<tr>
<td></td>
<td>libavfilter 3. 58.100 / 3. 58.100</td>
</tr>
<tr>
<td></td>
<td>libswscale 2. 2.100 / 2. 2.100</td>
</tr>
<tr>
<td></td>
<td>libswresample 0. 17.102 / 0. 17.102</td>
</tr>
<tr>
<td></td>
<td>libpostproc 52. 3.100 / 52. 3.100</td>
</tr>
<tr>
<td>Computer Vision platform</td>
<td>OpenCV x86_64 2.4.3-3</td>
</tr>
</tbody>
</table>
Chapter 3

Video Preprocessing

There are two types of characteristics of the video frames that are useful in tracking the bees:

**Tags:** Obviously, the tags are the best source of information regarding the movement of the bees. The actual coordinate of a bee, the orientation of its head and its unique ID can be determined just from decoding the tag.

**Edges:** Certain interactions between bees such as trophallaxis and touching of antennae cannot be inferred from the tags. Bodily protrusions of bees have clearly defined edges, which can be used in accurate measurement of antennae and tongue activity.

Preprocessing applied to the video should emphasize or at least preserve these two types of image characteristics.

3.1 Video compression

The sheer size of the video is a major limiting factor in the length of time bee movements can be recorded. For example, a movie file of around 11 hours and 15 minutes takes up around 870 GiB of disk space, not accounting for the file system overhead. Apart from storage, the movie data stream requires a large bandwidth when read, which limits the speed the video can be processed after acquisition, regardless of raw processing power [13].

3.1.1 Cropping

The original frame size is $1626 \times 1236$ pixels. It differs slightly from the actual capability of the camera ($1628 \times 1236$) in that the width of the former is not a multiple of 4. Video encoding software packages like Mencoder [14] discourage storing video data with frame width that is not a multiple of 4 since it interferes with word alignment in many recent processors [13] and requires excessive overhead in many popular compression schemes. Aside from the width problem, upper and lower parts of the video frame consistently display the wooden frame used to contain the bees, which is of no use in tracking the bees. For full codec compatibility, the frames were cropped to the lowest possible multiples of 16 in
both width and height in such a way as to not affect the area where bees can move.

### 3.1.2 Huffman YUV

The software product used to record the original video, Virtual VCR [15] only supported Huffman YUV compression [10]. Interestingly, the developer specification states that the frame width must be a multiple of 4 yet the software managed to bypass this limitation. The encoded format may not be supported by other decoders or players that comply to the HuffmanYUV standard.

The HuffmanYUV format uses intra-frame compression in that frames are processed independently of each other. The pixels values of a frame are scanned in sequence. A pixel at a particular location in this sequence is predicted using a simple heuristic, such as the median, applied to several of the preceding pixels. The difference between the actual and the predicted pixel value is compressed using Huffman coding [16]. This method is lossless in that the original uncompressed video can be reconstructed without error from the compressed format. The somewhat misleading term YUV refers to the fact that the codec requires that image data be stored in YCrCb format. RGB color space values R, G and B can be converted to Y, Cb and Cr by the following relation [17], assuming all intensities are represented by values in the interval $[0, 255]$:

\[
Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \\
Cb = -0.1687 \cdot R - 0.3313 \cdot G + 0.5 \cdot B + 128 \\
Cr = 0.5 \cdot R - 0.4187 \cdot G - 0.0813 \cdot B + 128
\] (3.1)

The extra chroma channels do not increase the file size considerably since they are always zero and always predicted accurately by most heuristics.

Nevertheless, for the 11 hour video file $d1_140812.avi$ mentioned previously, HuffmanYUV produces an average compressed frame size of around 1684 KiB out of an uncompressed size of 1965 KiB yielding an 85.7% compression ratio. The reduction in file size does not offset the compression/decompression overhead and the need for an index when seeking to a particular frame. Potentially lossy methods need to be employed in order to get the file size one order of magnitude lower to a level that makes recording of long video sequences feasible.

### 3.1.3 H.264/AVC

As of year 2013, the H.264 is the de facto format for high-quality video compression in the multimedia industry [12]. A reference implementation H.264 encoder is part of the SPEC2006 benchmarks, which are an integral part of the design process of CPU architectures such as the Intel Core [13]. H.264 is sometimes referred to as Advanced Video Coding (AVC) and H.264 and AVC can be used interchangeably.

H.264 encoding reduces file size by discarding information that is not perceived by the human visual system. Lossless compression is also possible by exploiting both spatial (intra-frame) and temporal (inter-frame) statistical redundancies.
Similar to the popular MPEG-2 and MPEG-4 standards, H.264 is based on the DPCM/DCT Video Codec Model. A diagram of a combined video encoding and decoding system is shown in figure 3.1.

![Figure 3.1: The DPCM/DCT Video Codec Model](image)

Compressed video frames can be grouped into three categories: I-frames, P-frames and B-frames.

I-frames (intra-coded frames) are compressed independently of other frames. Unlike in Huffman YUV, the pixels of an I-frame are grouped into non-overlapping square macroblocks (MB), usually ranging in size from \(4 \times 4\) to \(16 \times 16\) pixels. The macroblocks are encoded in a single pass over the frame from the top pixel rows to the bottom pixel rows. To exploit the similarities between the contents of neighboring MBs, the MB to be encoded is first estimated using an average (or one of many possible heuristics) of its neighbors that have already been encoded. Most often they are in the row(s) above or to the left. Blocks below and to the right cannot be used for prediction as they have not been seen by the encoder. Only the difference between the estimate and the actual block contents, also known as the *residual* is stored.

P-frames (predicted frames) are also split into P-frame macroblocks (PMBs). PMBs are estimated from one or more previous frame predictions using motion compensation. Previous frames need not be ahead in display order although they need to be ahead in the processing order of the coder/decoder (codec).

B-frames (bidirectional frames) can be made up of either P-frame or B-frame macroblocks (BMBs). BMBs are always motion estimated from frame predictions both ahead and after in display order. A BMB is equivalent to an interpolation between a past PMB and a future PMB, with the estimate relation being a (weighted) average or one of many heuristics.

The DPCM/DCT model describes in detail how P-frames are created. First, the raw image is divided into macroblocks in the same layout as an I-frame. For each macroblock, a search heuristic finds the macroblock in a previous frame that is the most similar to it. While the macroblock in the P-frame is aligned to a multiple of its size, the one in the previous frame need not be aligned. As a matter of fact, it can exceed the frame boundaries in which case
the part outside the frame is usually padded with zeroes. The displacement, called a motion vector, is stored. The collection of motion vectors is used to generate a motion compensated version of the P-frame. The difference, also called the residual, is transformed to the frequency domain using the Discrete Cosine Transform (DCT). Given a pixel intensity representation of the residual intensities $E = (e_{x,y})$, $x, y \in \{0, ..., N - 1\}$ where $N$ is the macroblock size, the frequency domain representation $F = (f_{i,j})$ is expressed as $F = A \times E \times A^T$ where:

$$
A_{x,y} = C_x \cdot \cos \left( \frac{(2 \cdot y + 1) \cdot x \cdot \pi}{2 \cdot N} \right) \quad \text{and} \quad C_x = \begin{cases} 
\sqrt{\frac{1}{N}} & \text{when } x = 0 \\
\sqrt{\frac{2}{N}} & \text{when } x > 0
\end{cases} \tag{3.2}
$$

The corresponding inverse transform iDCT is given by $E = A^T \times F \times A$.

The H.264 standard requires that the integer transform be used. It is an approximation of DCT with the added benefit that no information is lost through rounding to nearest integers.

The only step when information loss can occur is quantization. When a scalar quantizer is used, the integer frequency values $f_{x,y}$ are divided by a scalar called the quantizer or quantization parameter $QP$ yielding the quantized coefficients $q_{x,y}$:

$$
q_{x,y} = \text{round} \left( \frac{f_{x,y}}{QP} \right) \tag{3.3}
$$

If a $QP$ of 0 is specified, the encoder will skip this step ($q_{x,y} = f_{x,y}$) and create a bit stream from which the original video can be recreated without error.

Next, the coefficients are scanned, most often on a diagonal starting from the top-left corner, and turned into a data stream. The motion vectors computed in the motion estimation stage are difference coded and then concatenated to this stream. The entire resulting sequence is compressed in a lossless fashion using an entropy encoder. The size of the data is drastically reduced because the integer transform produces many small values, which are clamped to zero in the quantization stage. Like in Huffman coding, the entropy encoder produces longer symbols for less frequent values. However, it uses a predefined symbol table. Huffman coding needs to scan the data in order to compute the probability of occurrence for each symbol before starting the actual encoding process. The entropy encoder can process the data in a single pass and output symbols on the fly. It also takes advantage of the fact that the data stream contains long sequences of zero values by explicitly encoding run lengths of zero.

The symbol list is then stored in the video file. H.264 specifies only the encoding of the video data. How video and audio data are related, the index of the video frames for seeking, and other information is stored in the container, which has its own specification. In order to decode the video file, both the container and H.264 must be supported by the software system.

As mentioned previously, the motion estimation is based on the estimation of frames instead of original frames. This is to prevent the phenomenon called drift. If instead of the estimation, the raw frame would be used in the prediction, the errors in decompressing the residual would accumulate in time leading to an excessive loss of quality. Using the prediction instead keeps the error within bounds. In order to make predictions based on estimations, the encoder must also employ a decoder during compression.
Once the video data and the container are produced, they can be stored or streamed for later use.

The decoding process is the reverse of the encoding, where the opposite of quantization is rescaling:

$$\hat{f}_{x,y} = q_{x,y} \cdot QP$$

(3.4)

where $\hat{f}_{x,y}$ is the estimate of the original frequency coefficients $f_{i,j}$. The residual pixel intensity estimates $\hat{e}_{i,j}$ are obtained by iDCT from $\hat{f}_{i,j}$.

The motion compensation is then added to the residual to obtain the frame estimate $\hat{i}_{x,y}$.

The MSE quality measure

Knowing that the decoded frames are merely an approximation of the original ones, a measure of quality needs to be defined. This area has been the focus of extensive research. Measures that attempt at defining the video quality as perceived by humans include Just Noticeable Difference (JDD), Digital Video Quality (DVQ) and Structural Similarity Index (SSIM) to name a few [12].

In this work, compression constitutes a necessary preprocessing step for image analysis methodologies introduced in later sections. Information normally discarded by the human visual system may still be relevant and, as such, the simple and mathematically rigorous Mean Square Error (MSE) may be a better performance predictor for automated methods than those intended for subjective quality assessment. MSE is defined as the average pixel-by-pixel square difference between the original and predicted frames:

$$MSE = \frac{1}{WH} \cdot \left( \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} (\hat{i}_{x,y} - i_{x,y})^2 \right)$$

(3.5)

where $W$ and $H$ are the frame width and height, respectively.

x264

x264 [11] is a computationally efficient free and open-source H.264 encoding software package. It is employed in several web video services such as YouTube [18] and Vimeo [11]. The source code is optimized for the GCC compiler and Intel Core architecture, which makes it a good match for the computing environment utilized in this project.

YUV 4:2:0

x264 requires that the raw video frame be encoded in the YUV 4:2:0 format. Pixels are grouped in $2 \times 2$ pixel non-overlapping square regions. For each region, 4 luminance (Y) and one of each chroma values (Cr, Cb) are stored. This imposes the constraint that both the width and height of the frame be multiples of two. The luminance and chroma are stored separately, due to their different sampling frequencies. For the video files used in this project, the chroma values are both zero yet have to be explicitly specified when processed by x264. The entropy coding ensures that the chroma components have a negligible impact on file size.
3.1.4 x264 performance

In order to obtain an accurate estimate of the compression performance of x264, the entire 1 hour long video d1_140812.avi was compressed using fixed scalar quantizers ranging from 0 to 40 with an increment of 4. The YUV 4:2:0 color space used was yuvj420p instead of the default yuvj420p. The default color space maps several gray levels to a single value, leading to information loss.

On the whole video

The source video was read and the compressed video was written at the same time from the video file hard-disk unit. As it is evident from figure 3.2, despite the excessive disk usage, the encoding time was less than the duration of the source video (3600 sec) for all QP values tested. Therefore, faster than real-time encoding is possible. Lossless compression does not employ a quantization step so it encodes video faster than with quantizer 4. For low quantizers, motion estimation behaves in the same way as in lossless coding and quantization merely adds complexity to the encoding process while offering little file size reduction. As the quantizer increases, the compression time drops almost exponentially. One reason is that higher quantizers map many DCT coefficients to zero. Processing long runs of zero is faster and the entropy encoder produces fewer symbols. Also, disk reads and writes limit the performance of the encoding, regardless of CPU processing power. The smaller the compressed bit stream, the less data is written to disk shifting the compression burden from the disk to the CPU.

![x264 encoding time for a 3600 sec video](image)

Figure 3.2: x264 encoding time of a 1 hour long video

As shown in figure 3.3, lossy compression can dramatically reduce file size. While for low quantizers, the decrease is almost linear, from quantizer 20 onward file size drops exponentially. The gain in space does not seem to level off even
at the highest quantizer setting tested ($Q_P = 40$). At this point, the perceived quality was too low to warrant going further.

![Figure 3.3: Compressed size of a 1 hour long video](image)

Even lossless compression achieves a better than 2-fold size reduction (figure 3.4). Unlike Huffman YUV, which only exploits local spatial redundancy in video data, x264 in lossless mode eliminates both spatial and temporal redundancies across several frames.

When quantization is added, the performance increase is remarkable. While a quantizer of 4 offers little improvement over lossless compression, $Q_P$ values of 24 and 28 offer better than 50 fold and 100 fold file size reductions, respectively. Using a quantizer of 40, more than 600 times the space can be reduced.

If no motion prediction nor entropy encoding were to be employed, the file size ought to decrease logarithmically with the quantizer. Let the video sequence have $N = W \times H \times T$ pixels, where $W$, $H$ and $T$ are the frame width, height and total number of frames, respectively. A pixel can be one of 256 values giving a total uncompressed file size of $\log_2(256^N) = N \cdot 8$ bits. Through quantization, the number of symbols is reduced to approximately $256^{Q_P}$ leading to a file size of $\log_2 \left( \left( \frac{256}{Q_P} \right)^N \right) = N \cdot (8 - \log_2 Q_P)$.

The strength of H.264 lies in its ability to make accurate motion prediction owing to the variety of reference macroblocks available [12]. The residual contains so little information that many of its DCT components, particularly at high frequencies, are small enough to be reduced to zero for quantizers beyond 12. Entropy encoding is specially designed to handle long sequences of zero, explaining the sharp compression ratio increase for $Q_P \in [12, 20]$.

The importance of zero component elimination can be inferred also from the average MSE (figure 3.5). The error increase slows down from $Q_P = 16$ onward as the file size decreases at an accelerated pace. Evidently, the compression gains beyond this point from accurate predictions and not by removing information
from the residual.

Figure 3.4: x264 compression ratio

Figure 3.5: Average MSE over the whole 1 hour for various quantizers
**Frame by frame**

In order to understand how quality varies across the frames for very long sequences, frame by frame MSE values were computed for various quantizers across the entire hour long video.

Lossless compression did perform as expected, with $MSE = 0$ for every frame analyzed. In terms of frame mix, the 51481 frame sequence was encoded using 206 I-frames, 51275 P-frames and no B-frames.

For lossy compression, the frame quality variation is periodic as shown in figure 3.6. The frame sequence is split by the encoder into 250 frame groups of consecutive frames. The recommended framerate for H.264 is 25 fps, which means that each group was designed to span exactly 10 seconds. Each group appears to be encoded separately from the others. The 1000 frame MSEs plotted, show 4 of these groups.

![Figure 3.6: Long term MSE trend for popular quantizers](image)

The first frame of each group is an I-frame and has a distinctively low MSE. The frames that come after are either P or B-frames. This fact was deduced from the statistics outputted by the x264 encoder. Out of the 51481 frames of the hour long encoded video (at 14.3 fps), 206 are I-frames, 25722 are P-frames and 25553 are B-frames. Hence, every 250 group of frames has exactly one I-frame ($206 = \lceil \frac{51481}{250} \rceil$) while the rest are an even mix of P and B-frames.

The quantizer value range $[24, 32]$ is often used in practice [12] and the x264 encoder itself was designed for best performance around $QP = 26$ [11]. The MSE trend within the groups differs greatly across this range. For $QP = 24$, apart from the low error of the first frame (I-frame) in each group, the MSEs are
Quantizing the difference between the current frame and the estimate of the previously encoded frames prevents error accumulation or drift for these settings. For $QP = 32$, this balance breaks down and quality steadily degrades as the frames are farther away from the first (I-frame) in the group. A new group resets the MSE, which results in long term error stability. In this sense, the I-frames act as “fire-walls”, preventing errors from accumulating beyond them. Moreover, if the bit stream contained errors, I-frames ensure that at most 250 frames are affected. They are also useful in seeking at random points in video. To decompress any random frame, at most 250 frames need be read, starting with the I-frame of that group.

In the case of the hour-long video, the 264 encoder selected different quantizer values for the three types of frames. Given a global quantization parameter $QP$, I-frames are compressed with $QP - 3$, P-frames with $QP$ and B-frames with $QP + 2$. I-frames have no motion prediction and thus larger residuals. A lower quantizer is required to encode them accurately enough. Furthermore, the quality of the entire frame group depends on this frame and, since it is very infrequent, more space can be used to encode it. B-frames on the other hand rely on a great deal more motion compensation than any other frames meaning that their smaller residuals can withstand more quality loss.

Figure 3.7 shows a close-up view of the frames that lie at the boundary of two neighboring groups, for low quantizers. Frame A (the 10001st), being an I-frame, has a substantially lower MSE than the frames both preceding and following it. The MSE values of the P and B-frames oscillate in a saw-like pattern. Higher MSE values correspond to B-frames while lower to P-frames. From this plot, it can be inferred that a frame group is made up of the sequence $IBPB{PBP}{PB}...{BPB}PP$. This corresponds to the type 0 display order as described in the H.264 specification [12]. The frame types and their dependencies for this display order are shown in figure 3.8. P-frames are predicted based on the frame that is two time steps before it, be it another P-frame or an I-frame. A B-frame is expressed in terms of the frames immediately preceding and immediately following it. P-frames are stored one time step ahead and B-frames one time step later than in their display order. The position of I-frames is not altered. Before a B-frame is decoded, the succeeding P-frame needs to be decoded and buffered.

While the MSE values for P and B-frames alternate around a fixed level throughout a group, this no longer holds for higher quantizers. As it can be seen in figure 3.9, drift becomes a significant issue, with MSE values at the end of a group being much higher than at the beginning. MSE differences between neighboring frames are much smaller and consistently decrease with higher quantizer values.

MSE is a measure of the quantity of discarded information. It does not describe the nature of this information. The effect of compression on a representative P-frame (frame B) offers more insight (figure 3.10). The patch shows two bees mounted with tags, dripping sugar water and a tag that was ripped from the back of the bee and ended up in the hive frame. For $QP = 16$ (compression factor 8.8), the discarded information is Gaussian noise, most likely caused by the image acquisition system. No relevant detail in the original patch seems to have been removed. Compression at this level could be used as a noise reduction preprocessing method in itself.

For $QP = 28$ (compression factor 110), apart from noise, sharp transitions
Figure 3.7: MSE values over 30 representative frames for low quantizers

in the image are also affected. Most of these, like reflections off wings and liquid, actually interfere with automated analysis. Antennae and tags are slightly affected although bearing in mind that the image was emphasized by a factor of 5, it should not have a large impact on the performance of an automated system. The blurring at this quantization level should be comparable to or less prominent than motion blur caused by the long exposure time of the camera (70 ms) for active bees.

Once most noise has been removed, space can only be saved by discarding useful image information. In the case of $Q_P = 40$, high frequency components, including a great deal of edge information, are quantized to zero and irreversibly lost. Apart from irrelevant details such as hive edges and stripes on the bee abdomens, valuable information is also lost. The tag codes are mostly removed as are antennae and limbs. As it can be seen in figure 3.10(d), noise removal is no longer noticeable when compared to relevant detail loss.
Figure 3.8: The type 0 display order as defined by the H.264 standard. Transmission order refers to the order of frames in the compressed bit stream.

Figure 3.9: MSE values over a 250 frame chunk for high quantizers
Figure 3.10: The type of information that is discarded during the compression process, for various values of the scalar quantizer.
3.2 Background removal

The scenes are stationary with respect to the hive frame and the field of view has been cropped to encompass precisely the region where bees are allowed to move. Apart from the moving bees, the video images contain hexagonal hive cells and a small part of the wooden outer frame as background. The hive frame and outer frame remain mostly unaltered throughout the video sequence. In the future, bees may be filmed long enough to lay eggs and seal hive cells to keep sustenance and larvae. For the time being, however, the background is of least concern to this project.

At first, whether the background hinders the segmentation of the bees was studied.

Edge image

An edge image was obtained using an established method: the Canny edge detector \[19\]. This method was designed to identify true edges in the presence of noise, based on low pass filtering and connected components. First, the pixels in the original image \(i_{x,y}\) are smoothed using a Gaussian filter of standard deviation \(\sigma\) yielding the image \(f_{x,y}\). The kernel of the Gaussian filter is given by:

\[
G_{x,y} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}
\]  

(3.6)

The \(\sigma\) parameter controls minimum distance between two different edges. The gradients along the axes are computed as \(g_x = \frac{\partial f}{\partial x}\) and \(g_y = \frac{\partial f}{\partial y}\). Gradients can be computed by convolving with either [-1 0 1] or [-1 1] kernels and their transpositions. Next, the gradient magnitude \(M_{x,y}\) and direction \(\alpha_{x,y}\) are computed using:

\[
M_{x,y} = \sqrt{g_x^2 + g_y^2}
\]  

(3.7)

\[
\alpha_{x,y} = \begin{cases} 
\arctan\frac{g_y}{g_x} & \text{if } g_x \neq 0 \\
\text{sign}(g_y) \cdot \frac{\pi}{2} & \text{if } g_x = 0
\end{cases}
\]  

(3.8)

Magnitude tends to be high in non-edge pixels so nonmaxima suppression is used. For every pixel, the gradient direction \(\alpha_{x,y}\) is rounded to the closest multiple of \(\frac{\pi}{4}\), giving one of the 8 directions \(d_k\). If the magnitude of a pixel is not greater than the magnitude of either neighboring pixels along \(d_k\), it is set to zero. Intuitively, edges should be one pixel thick and nonmaxima suppression is a way of thinning the edges.

Two binary images \(l_{x,y}\) and \(h_{x,y}\) are obtained by thresholding the magnitude at every pixel with a low threshold \(T_L\) and a high threshold \(T_H\), respectively. The final edge image is the collection of pixels obtained by 8-neighborhood flood-filling \(l_{x,y}\) with seeds in \(h_{x,y}\). Edge pixels in \(l_{x,y}\) that are not reached by the flood-filling are removed. This is called hysteresis thresholding.

The utilized implementation, part of the OpenCV library \[20\], was designed for low computational cost and it differs from the original Canny method. Gaussian smoothing and gradient calculations were approximated with two Sobel filters \[21\]. The L2 norm in edge magnitude was approximated with the L1 norm as it does not require computing the square root.
Figure 3.11 shows the result of running this optimized version of Canny edge detector on a sample frame. Implementation details are summarized in table 3.1.

Table 3.1: Canny Edge Detector Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Threshold</td>
<td>100</td>
</tr>
<tr>
<td>High Threshold</td>
<td>200</td>
</tr>
<tr>
<td>Smoothing</td>
<td>Embedded in Sobel Operator</td>
</tr>
<tr>
<td>Sobel Operator Diameter</td>
<td>3</td>
</tr>
<tr>
<td>Gradient approximation</td>
<td>$G =</td>
</tr>
</tbody>
</table>

The antennae are picked up as edges and are hence outlined well, despite being difficult to discern in the original image. However, the hive cells have pronounced edges that are emphasized by the detector. Because the Canny method chooses the strongest edge within smoothing neighborhood, hive edges obstruct some relevant bee edges including antennae.

The local texture of the background hinders segmentation and it would be desirable to obtain a means to automatically remove the background or to replace it with a flat texture.
3.2.1 Clustering

Segmentation can be performed by classifying local texture as belonging to either background or foreground. The large variety of textures makes manual texture labeling tedious. If a large corpus of texture prototypes were to be generated automatically, dividing the corpus into bee and background sections would be a less daunting task.

In unsupervised learning, a system learns how to represent data based on a quality measure that is defined independently of the task at hand \[22\]. Apart from the measure, no manual input is necessary making it a viable preprocessing procedure for large sets of data.

Clustering is a form of unsupervised learning where data points are assigned groupings based on a similarity measure. One of the most simple and robust unsupervised clustering methods is K-means clustering \[23\]. It uses squared Euclidean distance as the similarity measure and the sum of intra-class variances as a data representation quality measure.

Texture based segmentation requires that a mathematical model of texture be specified. For simplicity, local texture is defined in this work as a pixel-wise transform of a square patch centered at a particular location \((x, y)\) as \(\vec{t}_{x,y} = f(i, x', y')\) where \(x', y' \in \{-r, ..., r\}\), \(r\) is the (Manhattan distance circle) radius of that patch and \(f\) is the transform, often a normalization function.

The textures can be scanned from top-down and then left-right making them one-dimensional vectors \(\vec{t}_{x,y} = (v_{j})\) where \(j \in \{0, ..., m-1\}\) is an integer. The length of the vector is given by \(m = (2 \cdot r + 1)^2\). The textures themselves can be serialized as to not depend to their center coordinates and are indexed as \(t_i\), where \(i \in \{0, ..., n-1\}\) (not to be confused with \(i_{x,y}\)) and \(n\) is the total number of texture samples gathered. An individual texture value can be expressed as \(v_{i,j}\) where \(i\) is the texture index and \(j\) is the location within the texture.

Clusters are defined as the partition of the set \(\{0, ..., n-1\}\) into mutually disjoint subsets \(C_k\) that minimizes the objective function \(J(C)\). The total number of clusters \(K\) is decided beforehand. The objective function is given by:

\[
J(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{i, i' \in C_k} d(\vec{t}_i, \vec{t}_{i'})
\]  

(3.9)

where \(d\) is the square of the Euclidean distance:

\[
d(\vec{t}_i, \vec{t}_{i'}) = \sum_{j, j' \in \{0, ..., m-1\}} (v_{i,j} - v_{i',j'})^2
\]  

(3.10)

The advantage of K-means is in that the objective function reduces to:

\[
J(C) = \sum_{k=1}^{K} \sum_{i \in C_k} d \left( \vec{t}_i - \hat{\mu}_k \right)
\]  

(3.11)

where \(\hat{\mu}_k\) is the centroid of \(C_k\), the average of all textures assigned to \(C_k\). Unlike other learning systems, K-means is transparent in that \(\hat{\mu}_k\) is a texture that defines its cluster (a prototype).

The calculation of the prototypes is accomplished through an iterative descent algorithm:
1. Start with a number of clusters \( K \) and a value for each \( \hat{\mu}_k \) computed based on some heuristic on the data.

2. For each \( k \), compute \( C_k \) as the set of indices \( i \) such that \( \vec{t}_i \) is closer to \( \hat{\mu}_k \) than any other \( \hat{\mu}_{k'} \), \( k \neq k' \).

3. Update for cluster \( \hat{\mu}_k \) to be the average of all \( \vec{t}_i \), \( i \in C_k \).

4. If no \( \hat{\mu}_k \) has changed in this iteration or a certain number of iterations has been reached, terminate the algorithm. Otherwise go back to step 2.

Since the number of clusters is to be specified a priori and greatly affects the outcome, a large number of clusters was chosen. The initial cluster positions were chosen as random patches from a video frame. As the illumination and the structure of the frames changes little throughout the sequence, it is assumed that it is very likely that patches from one frame can be also found in other frames. The high number of patches also means that the centroids span the texture space evenly with high probability.

Shadows may induce unwanted variance and can be alleviated by normalization. The simplest way is to bring all pixel values of the neighborhood into the same fixed range. This can be accomplished through min-max normalization. The pixel values can be linearly mapped to span the interval \([0, 255]\). Let \( \min_{x,y} \) and \( \max_{x,y} \) be the minimum and maximum values of the image patch centered at \((x, y)\):

\[
\min_{x,y} = \min_{x',y' \in \{-r,\ldots,r\}} i_{x+x',y+y'} \quad (3.12)
\]

\[
\max_{x,y} = \max_{x',y' \in \{-r,\ldots,r\}} i_{x+x',y+y'} \quad (3.13)
\]

The normalization function \( f \) is:

\[
f(i_{x,y}) = 255 \cdot \frac{i_{x,y} - \min_{x,y}}{\max_{x,y} - \min_{x,y}} \quad (3.14)
\]

Another normalization method is Gaussian normalization. Let \( \mu_{x,y} \) and \( \sigma_{x,y} \) be the mean and standard deviation of the image patch centered at \((x, y)\):

\[
\mu_{x,y} = \frac{1}{(2 \cdot r + 1)^2} \cdot \sum_{x',y' \in \{-r,\ldots,r\}} i_{x+x',y+y'} \quad (3.15)
\]

\[
\sigma_{x,y} = \sqrt{\frac{\sum_{x',y' \in \{-r,\ldots,r\}} (i_{x+x',y+y'} - \mu_{x,y})^2}{(2 \cdot r + 1)^2 - 1}} \quad (3.16)
\]

giving a normalization function

\[
f(i_{x,y}) = \text{saturate} \left( 128 + 128 \cdot \frac{i_{x,y} - \mu_{x,y}}{\sigma_{x,y}} \right) \quad (3.17)
\]

where the values are saturated to the interval \([0, 255]\) by:

\[
\text{saturate}(x) = \begin{cases} 
0 & \text{if } x < 0 \\
x & \text{if } 0 \leq x \leq 255 \\
255 & \text{if } x > 255 
\end{cases} \quad (3.18)
\]
Gaussian normalization is robust even in the presence of shot noise in the imaging system or glares caused by dripping liquid. Nonetheless, it cannot guarantee that no pixel values get perturbed by saturation nor that the \([0, 255]\) range can be covered efficiently.

The centroid (prototype) textures for 64 clusters obtained through each normalization technique are shown in figure 3.12.

![Textures](image)

(a) No normalization. Textures sorted by average value  
(b) Gaussian Normalization  
(c) Min-max Normalization

Figure 3.12: Results of running K-means clustering on a typical frame (frame A) using 11 x 11 pixel patches and 64 clusters

Because the hive cells are regular, the space of all possible cell textures is small. Also, most of the video frame is made up of cells and the initial cluster seeding was made with equal probability over the image. Most clusters are created to represent cells. The combined effect is that the cell texture prototypes converge to strong, regular, well defined features. Bees are highly irregular and regions of the video frame containing bees have fewer cluster assigned to them. The clusters that describe bee regions end up covering heterogeneous regions in texture space and thus the prototypes are averaged away into smooth gradients.

In particular for small neighborhoods, deciding which clusters belong to bees and which to background can be challenging. Larger neighborhoods require a large number of clusters to be created in order to span a high dimensional texture space. In this case, manually selecting which clusters belong to bees and which to background is tedious. To facilitate the selection, regions in frame A were
manually marked with red for definitely background and green for definitely bee. Areas of the image were left unpainted as it was difficult to establish which pixels belonged to bees and which to background in the vicinity of bee bodies and under shadows. This human editable image was transformed into a single channel image as in figure 3.13. The gray valued pixels are considered unlabeled and not used in training or testing.

Figure 3.13: Training data for supervised post-processing

Next, the nearest cluster for each pixel in the frame image was computed. As a result, each cluster has been assigned a collection of pixels. By replacing the pixels with their corresponding labels, each cluster thus has a positive label count and negative label count. Ideally, each cluster should have one of these counts equal to zero. In practice, each cluster has a pixel classification error, or misclassification impurity.

Since the number of positive labels differs from the negative labels, the counts need to be normalized using:

\[
P_f = \frac{\text{count(foreground)}}{\text{prior\_count(foreground)}} \quad P_b = \frac{\text{count(background)}}{\text{prior\_count(background)}}
\]

(3.19)

to account for the prior imbalance and

\[
p_f = \frac{P_f}{P_f + P_b} \quad p_b = \frac{P_b}{P_f + P_b}
\]

(3.20)
to make sure that the probabilities sum up to 1.

There are several measures of impurity, the most common being:

- Misclassification Impurity: \( M_i = 1 - \max(p_f, p_b) \)
- Gini impurity: \( G_i = 2 \cdot p_f \cdot p_b \)
• Entropy impurity: \[ E_i = -p_f \cdot \log_2(p_f) - p_b \cdot \log_2(p_b) \]

Min-max normalization gave the lowest overall impurity values and these are shown in figure 3.14. The clusters were sorted by the normalized positive probability. Regardless of the measure used, cluster impurity remains within the 20% - 80% range.

![Figure 3.14: Classification purity on training image](image)

The clusters are assigned labels, either positive, or negative based on which normalized probability is greater. The label assignment is discrete, not fuzzy, and the purity measure is not taken into account. Because each pixel in the image can have the closest cluster assigned to it and the clusters are classified themselves into two classes, a binary classification of the pixels themselves is possible. Figure 3.15 shows the classification of the training image itself.

Hive cell edges are correctly classified as background. Most parts of bees are also classified correctly despite the fact that only a few of them were marked in the training process. The hive cell centers, having a more even texture are incorrectly marked as belonging to bees. If the background regions were to be removed and in-painted using the surrounding colors, edge detectors should not pick up hive cells while still focusing on antennae.

Normalized convolution was chosen as the inpainting procedure for its simplicity and speed of execution. This procedure takes two parameters: the original image \( i_{x,y} \) and a mask \( b_{x,y} \) that specifies which pixels are to be inpainted.

First, an inverse mask, \( f_{x,y} = 255 - b_{x,y} \), is computed to designate which pixels are to be left unchanged. The original image is masked with \( f_{x,y} \) to yield an image where all pixels to be estimated are set to black: \( f m_{x,y} \). This image is blurred using a Gaussian filter yielding \( f g_{x,y} \). The \( \sigma \) parameter should be large enough for the filter to fill in all the black pixels. Then the mask \( f_{x,y} \) is blurred with the same Gaussian filter to give \( m g_{x,y} \). The ratio between \( f g_{x,y} \) and \( m g_{x,y} \) is used to fill in the original image at the pixels designated by the
mask \( b_{x,y} \) producing the desired inpainted image: \( ip_{x,y} \). The whole process can be summed up as:

\[
ip = \left( \frac{(i \land m) \otimes G_\sigma}{m \otimes G_\sigma} \land m \right) \lor (i \land b)
\]  

(3.21)

The results of the inpainting procedure are shown in figure 3.16. Tags, having strong edges of orientations similar to the hive edges are misclassified as background. The bee and most importantly antennae outlines are however preserved by this procedure. Running the Canny edge detection with the same parameters as in table 3.1 yielded the image in figure 3.17. The detector managed to pick up the outlines of isolated bees and their antennae. Thus, background removal through clustering followed by inpainting may be utilized as a preprocessing stage of an edge-based bee detector or classification system. The design of such a system, due to its complexity, is beyond the scope of this work and is left as an open avenue for future research.

So far the fact that the frames are in temporal sequence has not been taken into account. Also, the background remains stationary throughout the videos and this can be used to estimate it. In the following section, motion based background estimation and removal are explored.

Figure 3.15: Cluster segmentation on the training frame A
Figure 3.16: Frame D after inpainting the background regions using normalized convolution (sigma = 2.0)

Figure 3.17: The effect of the K-means edge classification method on frame D. Green denotes edges found by the Canny method in the inpainted image while red represents the edges of the original frame D. Running K-means clustering only on edge pixels of the original image produces the same result with greatly reduced computation time.
3.2.2 Frame differencing

The most simple motion detection method is computing the arithmetic difference between the corresponding intensity values of consecutive frames [24]. When an object moves over a stationary background, the frame difference should have most background values equal to zero. As it can be seen in figure 3.18, bees are darker than the hive so areas where the bees move into are dark in the frame difference while previous positions are white. If a bee remains motionless even for a few frames, it disappears completely from the difference frame. Many bees wiggle around a fixed position and as a result they are barely visible. This method is not effective in tracking slow moving bees.

![Figure 3.18: The difference between two consecutive frames, emphasized by a factor of 5](image)

When two individuals interact, their antennae move fast enough to be present in the difference frame. Both current and previous positions are clearly outlined. A line segment detector could be employed to pick up these positions and record interactions, although the actual methodology is left for future research.

Only bee outlines are visible in the difference frame and the actual locations of the tags are not revealed. A system with memory over a longer term is needed to mark the entire bee body and not just recent motion.

3.2.3 Exponentially weighted moving average

Exponentially weighted moving average (EWMA), successfully applied in system control [25], estimates the background over a longer sequence of frames. In this technique, a sample’s influence on the present decreases exponentially with age. The background estimate at time $t$, $b_{x,y,t}$, can be expressed in terms of the image gray levels at various times $i_{x,y,t'}$ as:
\[
\hat{b}_{x,y,t} = \frac{\sum_{t'=0}^{\infty} (\beta^{t'} \cdot i_{x,y,t-t'})}{\sum_{t'=0}^{\infty} \beta^{t'}}
\]  
(3.22)

where \(0 < \beta < 1\) is a parameter that controls how fast the weights decay. The total sum of the weights has to be 1 in order to prevent the background estimate either vanishing to zero or growing out of control. By noting that the denominator is equal to \(\frac{1}{1-\beta}\), the equation can be simplified with the substitution \(\beta = 1 - \alpha\):

\[
\hat{b}_{x,y,t} = \alpha \cdot (\sum_{t'=0}^{\infty} (1-\alpha)^{t'} \cdot i_{x,y,t-t'})
\]  
(3.23)

where \(\alpha\) is the learning rate, a parameter that controls how far in the past frames have a non-negligible influence on the estimate. Equation 3.23 is equivalent to the following recursive relation:

\[
\hat{b}_{x,y,t} = \alpha \cdot i_{x,y,t} + (1-\alpha) \cdot \hat{b}_{x,y,t-1}
\]  
(3.24)

The estimate can be updated in-place using only the current frame data, making EWMA very light in terms of both memory usage and computational load. A minor drawback is that the background estimate has to be kept in floating point representation.

The background estimate and its removal are highly dependent on the value of the learning rate and the background initialization. In this work, the estimate is at the beginning set to be the first frame processed, which does include the bees (foreground) as well. The result of EWMA background removal on frame A (the 10001st) for a large learning rate of \(\alpha = 0.01\) is shown in figure 3.19. The background estimate has been accumulated over 10000 frames. The weight of the initial background value is \(\alpha \cdot (1-\alpha)^{10000} = 2.25 \cdot 10^{-46}\), which is negligible. The initialization does not affect the estimate this far away in time although recent frames do have a large weight. As a consequence, trails can be seen where bees were recently moving as there is a delay until recent foreground values are eliminated from the background estimate.

Smaller learning rates result in a more equal weighting of past samples and a more robust background estimate. A drawback is that the background needs to accumulate over a very large number of frames in order to downplay the role of the initialization. Figure 3.20 shows frame A with the background removed after background accumulation over 10000 frames. The weights are spread out more evenly. The previous frame has a weight of \(\alpha \cdot (1-\alpha)^1 = 0.99 \cdot 10^{-4}\) while the first frame analyzed has a weight of \(\alpha \cdot (1-\alpha)^{10000} = 0.37 \cdot 10^{-4}\). As the initial estimate and following frames are difficult to eliminate, the initial bee positions linger as white blobs in the background removed frame.
Figure 3.19: Frame A after EWMA background removal, emphasized by a factor of 2. The background has been accumulated with a learning rate of $\alpha = 0.01$ over 10000 frames.

Figure 3.20: Frame A after EWMA background removal, emphasized by a factor of 2. The background has been accumulated with learning rate of $\alpha = 0.0001$ over 10000 frames.
Temporal histogram

It was observed that even in areas where no motion occurs, gray levels oscillate slightly in time. This is most likely due to small errors inherent in the camera system. In order to estimate the background, the temporal histograms of the gray levels for each pixel, over the span of 1 hour (51480 frames), were computed. The four distinct types of histograms encountered are summarized in figure 3.21.

Figure 3.21: Major types of temporal histograms

1. For a pixel far apart from the areas where bees are generally present, the histogram has a single peak with a value corresponding to the actual gray value of the background. Variations from this peak are caused both by atmospheric effects and errors in the imaging system. Nevertheless, the histogram shape is approximated well by a Gaussian distribution, whose properties can be inferred through regression.

2. The pixel inside a moderately crowded area of the swarm shows several peaks. The rightmost corresponds to the background since the background is the brightest object in the scene. The second peak corresponds to bee shadows. It is much flatter and less defined because shadows vary greatly in intensity depending on the positions of the surrounding bees. The leftmost peak belongs to the actual bee gray values. Even more so than with shadows, bees have a complex texture spanning a wide range of gray levels. Hence this peak has a more irregular shape.

3. In a more crowded area of the image, the peaks stand for the same three parts of the image. The tallest peak belongs to shadows, which means that bees tend to linger there most of the time and cast long shadows on the hive cells.

4. In one of the most crowded areas of the frame, the background peak is absent since at almost all times that area is occupied either by a bee or
its shadow. Estimating the background from the temporal histogram is the most difficult in this area.

In order to simplify the analysis of the background, several assumptions are made in the beginning. These assumptions will be relaxed later on to yield increasingly complex models of the background. At first, it is assumed that:

1. The background can be considered an image, which does not change during the span of time the histograms were computed.

2. All histograms can be approximated by a single Gaussian distribution.

3. The proportion of non-background gray levels in every temporal histogram is negligible.

4. The background value is given by the mean of the Gaussian distribution. Even when the mean corresponds to a bee gray level, contrast can be increased in this area to a level that can compensate for the background removal.

5. The variance of each individual Gaussian distribution is irrelevant since only the mean is used.

6. The mean of every Gaussian distribution is given by a simple function computed over the temporal gray-level histogram.

3.2.4 True average

First, it is assumed that the mean of the Gaussian distribution is approximated well by the average of the gray values over the entire time span the histograms were computed. Given these assumptions, the estimate of the background image \( \hat{b}_{x,y} \) can be computed directly from the temporal histograms \( h_{x,y}(i) \) as:

\[
\hat{b}_{x,y} = \frac{\sum_{i=0}^{255} (h_{x,y}(i) \cdot i)}{\sum_{i=0}^{255} h_{x,y}(i)}
\]  

(3.25)

Here, all frames have equal influence on the background estimate. If the foreground pixels are evenly distributed both above and below background values, the average will successfully eliminate them.

3.2.5 Histogram peak

Assuming that the mean of the Gaussian distribution corresponds to the highest histogram count, the background image estimate is given by:

\[
\hat{b}_{x,y} = \arg \max_{i \in \{0,...,255\}} (h_{x,y}(i))
\]  

(3.26)

A comparison between a patch in the average estimate and this peak estimate is shown in fig. 3.22. The peaks are susceptible to noise in the data, which
introduces sharp transitions in the estimate. Shadows contrast strongly with brightly lit regions. Subtracting this estimate from the frames would induce artificial edges, which may interfere with edge-based segmentation methods. A more robust method is needed to prevent this from happening.

![Image](a) Part of an average background estimate. (b) Corresponding part in the histogram peak background estimate.

Figure 3.22: Discontinuities in the histogram peaks image alongside a smooth estimation like the average.

### 3.2.6 Percentile

By relaxing the assumption that the foreground influence on the histogram is negligible, a percentile can be specified as a parameter. For example, if the foreground pixels were to correspond to the lower 20% of the gray values in the monitored sequence, the remaining 80% upper portion can be considered background and bell shaped so the background image can be approximated as the $20 + \frac{80}{2} = 60^{th}$ percentile. The background image can be computed efficiently using only the temporal histogram. Given a percentile $p \in (0,1)$, the estimate of the background can be expressed in terms of the cumulative histogram $c_{x,y}$, defined as:

$$c_{x,y}(i) = \sum_{j=0}^{i} h_{x,y}(j)$$  \hspace{2cm} (3.27)

yielding a background estimate that is the $p^{th}$ percentile of the cumulative histogram:

$$\hat{b}_{x,y} = \text{i. s. t.} \begin{cases} c_{x,y}(i) \geq p \cdot c_{x,y}(255) \\ c_{x,y}(i + 1) < p \cdot c_{x,y}(255) \end{cases}$$ \hspace{2cm} (3.28)

Here $c_{x,y}(255)$ is the sum of all the histogram counts.

Various background estimations were obtained for different values of $p$. The best estimate was empirically found to correspond to the median, or the $50^{th}$ percentile. It seems that the assumption that the foreground distribution is negligible is valid, as supported by [26]. The median background estimate is shown in fig. 3.23.

Unlike the histogram peak method, the estimate does not present artificial edges. It is very similar to the one obtained by averaging although the median, being a statistical estimator, has an inherent robustness to outliers. Figure 3.24 shows a median background patch and its difference from the corresponding average estimate. The discrepancy is hardly noticeable though the average does
incorporate more foreground values (bees and shadows) that end up as small smudges in the background estimate. The median gives gray values weight based on how often they occur, emphasizing objects with lower gray level variance. The variance of the hive frame is given by imaging noise, which is still lower than that caused by bee motion. Thus, the median gives a more accurate estimation of the background than the average.

Figure 3.23: Median (50th percentile) pixel values over 1 hour

(a) Part of a median background estimate
(b) Corresponding part in the average background estimate
(c) Difference between corresponding parts in the average and median estimates emphasized by a factor of 5

Figure 3.24: Advantage of the median estimate over the average. Median removes traces of motion that are incorporated in the average
3.2.7 Mixture of Gaussians

Regular variations in the background such as shadows, can be accounted for by relaxing the constraint that histograms take the form of a single Gaussian distribution. As stated in temporal histogram analysis, the temporal histogram can be considered to be made up of 3 Gaussian distributions: regular background, shadows and bees.

This approximation can be computed online (at the same time as the frames are acquired) using the mixture of Gaussians (MOG) method [27]. In general, for every pixel in the frame a list of K Gaussian distributions is maintained. Each distribution \( k \in \{0, \ldots, K - 1\} \) has two parameters: the mean \( \mu_{k,t} \) and the variance \( \sigma_{k,t} \). In addition, two more values are maintained: a weight \( w_{k,t} \) and a background confidence value equal to \( \frac{w_{k,t}}{\sigma_{k,t}} \).

Given this model, the probability that the current gray level is observed is given by:

\[
P(i_{x,y,t}) = \sum_{k=1}^{K} (w_{k,t} \cdot \eta(i_{x,y,t}, \mu_{k,t}, \sigma_{k,t}))
\]

(3.29)

where \( \eta(i_{x,y,t}, \mu_{k,t}, \sigma_{k,t}) \) is the Gaussian probability density function of the \( k \)th Gaussian distribution and is given by:

\[
\eta(i_{x,y,t}, \mu_{k,t}, \sigma_{k,t}) = \frac{1}{\sqrt{2 \cdot \pi \cdot \sigma_{k,t}}} \cdot \exp \left( -\frac{(i_{x,y,t} - \mu_{k,t})^2}{2 \cdot \sigma_{k,t}^2} \right)
\]

(3.30)

If the gray levels are observed over the 1 hour sequence, then, the probability density function \( P \) is actually given by the normalized histogram:

\[
P(i_{x,y}) = \frac{h_{x,y}(i_{x,y})}{255 \sum_{i'=0}^{255} h_{x,y}(i')}
\]

(3.31)

The weights \( w_{k,t} \) express how frequent values belonging to the \( k \)th distribution are found at that position in the frame. Background gray values occur frequently (high weight) and do not span a wide range (low variance). Whether a Gaussian distribution corresponds to background or not can be estimated by checking whether that distribution has a high confidence value \( \frac{w_{k,t}}{\sigma_{k,t}} \).

Maintaining the temporal histogram and obtaining the Gaussian mixture parameters by regression at every time step is too memory consuming and computationally intensive a task to be performed online. Instead a simple search is performed. At every time step, the distributions are sorted by confidence value so that the ones that are most likely to be background occur first. Then, for \( k \) increasing from 0, it is checked whether the new gray value \( i_{x,y,t} \) falls within \( 2.5 \cdot \sigma_{k,t-1} \). If a match is found, the search stops. Often, the distance between the means of two Gaussian distributions is less than \( 5 \cdot \sigma_{k,t-1} \), meaning that they overlap. The MOG method gives precedence to background distributions.

If no match is found, the distribution most likely to be foreground is replaced with a distribution centered at this gray value with a high variance and low weight. Otherwise, the parameters of the matching distribution \( k \) are updated:

- The weight is brought closer to a value of 1:
where $\alpha$ is a constant called the learning rate. It controls the speed the weights approach either 1 or 0 and thus the minimum velocity an object needs to have in order to be classified as foreground.

- The mean is brought nearer to the current gray value:

$$\mu_{k,t} = (1 - \rho_{k,t}) \cdot \mu_{k,t-1} + \rho_{k,t} \cdot i_{x,y,t} \quad (3.33)$$

where $\rho_{k,t}$ is the second learning rate that is dynamically updated as:

$$\rho_{k,t} = \alpha \cdot \eta(i_{x,y,t}, \mu_{k,t}, \sigma_{k,t}) \quad (3.34)$$

- The variance is updated to incorporate the distance between the new gray value and the mean:

$$\sigma^2_{k,t} = (1 - \rho_{k,t}) \cdot \sigma^2_{k,t-1} + \rho_{k,t} \cdot (i_{x,y,t} - \mu_{k,t})^2 \quad (3.35)$$

The parameters of the non-matching distributions remain unaltered with the exception of the weight, which is decreased exponentially towards 0:

$$w_{k,t} = (1 - \alpha) \cdot w_{k,t-1} \quad (3.36)$$

Computing the second learning rate for every pixel of the frame at every time step is computationally expensive. The OpenCV implementation of the MOG method that was used in conducting the experiments was found to make several simplifying assumptions. First the second learning rate is set to be $\alpha$ and thus a constant. This is allowed in the original method specification [27]. Multiplications are considered more expensive than additions and subtractions and the update rules are refactored to contain as few multiplications as possible.

The weights of non-matching distributions are left unaltered. The weights do not diverge towards infinity and are actually attracted to the $[0, 1]$ range.

The parameter updated rules become:

$$w_{k,t} = w_{k,t-1} + \alpha \cdot (1 - w_{k,t-1}) \quad (3.37)$$

$$\mu_{k,t} = \mu_{k,t-1} + \alpha \cdot (i_{x,y,t} - \mu_{k,t-1}) \quad (3.38)$$

$$\sigma^2_{k,t} = \sigma^2_{k,t-1} + \alpha \cdot ((i_{x,y,t} - \mu_{k,t-1})^2 - \sigma^2_{k,t-1}) \quad (3.39)$$

The parameter $T$ specifies the ratio between foreground and background in the average frame. The first $B_t$ Gaussian distributions whose cumulative normalized weights are no less than $T$ are considered background while the rest are foreground. $B_t$ can be expressed mathematically as follows:
\[ B_t = \arg \min_{b \in \{0, \ldots, K-1\}} \left( \frac{\sum_{k=1}^{b} w_{k,t}}{\sum_{k=1}^{K} w_{k,t}} > T \right) \] (3.40)

Experiments were conducted with \( K = 3 \) Gaussian distributions and the parameters listed in table 3.2, which yielded the best results. In the beginning, all mixtures are marked as foreground so the first Gaussian mixture will end up being the first frame, foreground included.

The second Gaussian is initialized gradually with gray values that do not fit the first distribution. An example is shown, after being updated for 100 frames, in figure 3.25. Evidently, updates are based on motion and the trails left by the bees accumulate over time. It is difficult to discern whether the mean of this distribution corresponds to shadows, bee bodies or both. It is definitely distinct from the hive frame.

The foreground pixels extracted from frame \( A \) are shown in figure 3.26. The Gaussian mixture was updated over 10000 frames in order to minimize the effects of the initialization. Fast moving bees are well segmented though the majority do not move fast enough to be labeled as foreground. Interestingly, the white areas of the tags are included in the foreground as they are more similar in intensity with the hive frame underneath. The drawback of choosing a global constant second learning rate becomes apparent. Shadows move as fast as the bees projecting them while there is a large variation in bee movement speed. It is difficult to distinguish between shadows and bees based only on gray level variance.

<table>
<thead>
<tr>
<th>Table 3.2: MOG Experiment Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of distributions</td>
</tr>
<tr>
<td>Learning rate (( \alpha ))</td>
</tr>
<tr>
<td>Background ratio (( T ))</td>
</tr>
<tr>
<td>Default initial (low) weight</td>
</tr>
<tr>
<td>Minimum variance (noise variance)</td>
</tr>
<tr>
<td>Default initial (high) variance</td>
</tr>
</tbody>
</table>
Figure 3.25: The second Gaussian mean after being updated for 100 frames

Figure 3.26: Foreground pixels of frame A after learning the background mixtures over 10000 frames
3.3 Discussion and future work

Clustering seems to be effective in removing background edge information. Combined with an edge detector, it can form a powerful edge classification technique. The texture prototypes are also human readable and the entire process is "open" making it easier to predict performance on unseen data. Clusters influence the area around them making this method similar to radial basis function neural networks (RBFNN) [22]. In a manner similar to RBFNNs, the clusters could be updated online to adapt to changes in lighting, for example.

The major drawback of clustering is its high computational complexity. The texture prototypes themselves only represent background edges so unsupervised learning cannot infer texture patterns representative of all the objects seen in the video. By marking clusters to be either foreground or background, a binary segmentation can be achieved. A major drawback is that the computational complexity increases with the number of clusters.

The cluster boundaries form a Voronoi diagram [28]. Grouping clustering into two classes create a complex decision boundary made up of numerous hyperplane sections. Should this boundary be continuous, or at least be made up of a small number of continuous regions, the boundary could be estimated by linear or non-linear regression. In effect, the cluster based texture classification could be approximated with a committee of perceptrons. Since Voronoi edges are distanced as far apart from cluster centers as possible, maximum margin Support Vector Machines [29], which are discussed in more detail in later sections, could provide a more efficient substitute.

Even with this simplification, all possible neighborhoods, approximately as many as the number of pixels in every frame have to be processed in order to obtain a background filtered image. If only edge classification is necessary, then only edge pixels in the original frame need to be tested, greatly increasing processing speed. More work needs to be done to determine whether the edge classification using K-means clustering is accurate enough to be used in interaction detection.

Frame differencing is a simple and fast method for detecting rapid motion. The shapes of antennae and possibly other protrusions are preserved. Most noise and some parts of bee bodies are not removed and may interfere with a subsequent antennae detection phase. Noise can filtered out by simple thresholding. A threshold value of around 2 times the standard deviation that, based on the observation that the noise is Gaussian, should filter out about 95% of the noise. A shape descriptor is required to separate bee motion from antennae that should be not very difficult to implement given that antennae are elongated and of constant width. Prior knowledge on antennae movement should be used to increase accuracy. The actual implementation of an antennae detector was not included in this work and is left as on open avenue for research.

Exponentially weighted moving average suffers from its reliance on a global learning rate. Low rates require a very long time for an accurate background estimation to accumulate. For higher rates, slow moving bees end up assimilated into the background.

Of the offline techniques, the median background estimate proved to be the most accurate. The median background removal does require two passes over the video: the first to compute the histogram and a second for perform the actual removal. This limits the analysis to stored videos. The H.264 encoder
manages to reduce the file size by an order of magnitude while preserving the relevant details making offline methods practical. The one drawback, which is difficult to alleviate is the effect of long term changes in the background such as bees dying in the middle of the video.

The results of the mixture of Gaussians method were disappointing. The approximations made to speed up processing had a detrimental effect on foreground estimation. The MOG method was designed primarily to deal with repeated variations in background. The bee shadow motions are closely linked to bee motion and shadows and bee bodies appear to have overlapping gray levels. Offline sum of 3 Gaussians regression of the temporal histogram may prove more effective though it should take into account that some distributions like background or bees are absent from certain parts of the video.

In areas of the frame where the hive cells are almost always in shadow, it may be more accurate to define the shadow itself as background and subtract it from the image. The median estimator seems to be the most suitable background removal method and it was chosen as the preprocessing step in the following section, which focuses on tag detection.
Chapter 4

Tag detection

4.1 Why is tag detection difficult?

Up to this point, methods for removing or reducing the influence of the background have been explored with the goal of discarding irrelevant information while preserving and perhaps enhancing relevant details. While edge-based analysis is left for open for future research, tag detection is treated in depth in this section.

In the 1 hour sequence studied, lighting is uneven across the frame. The imaging system limits the number of gray levels to 256. Depending on the area of the frame that is processed gray level occupy an even more limited range. Fig. 4.1(a) shows the appearance of a tag in a well lit region.

![Tag images and histograms](image)

Figure 4.1: Tag appearance in video

Its histogram spans over two thirds of the 256 level range. It can be easily
identified and encoded by a human observer. On the other hand, the gray level range of a tag under shadow, as shown in figure 4.1(b), is less than \( \frac{1}{5} \)th the maximum range. It is barely visible to the human eye and extremely difficult to decode. The tag in shadow can be lighting and contrast enhanced to span the entire 256 level range yielding the image in figure 4.1(c). It can now be identified though decoding is still difficult since in the low contrast original different digits mapped to the same gray levels. Furthermore, if this method were employed in tag detection, over a million candidate regions would have to be lighting and contrast enhanced to make sure tags in shadow are detected. Such naive preprocessing is too expensive to allow for real-time detection and more efficient methods are explored.

Confusion matrix and accuracy measures

In order to evaluate a tag detection method, reliable accuracy measures have to be defined. For example, it can be considered that almost every pixel in a frame (with the exception of a border equal to half the width of the tag), is a potential tag center. The candidate count would thus exceed 1 million while the actual number of tags is less than 200. If the naive definition of accuracy:

\[
\text{accuracy} = \frac{\text{number of correctly classified pixels}}{\text{total number of candidate pixels}}
\]  

(4.1)

were to be used, a trivial algorithm that classifies all pixels as not being the center of a tag would have an accuracy greater than \( \frac{1,000,000 - 200}{1,000,000} = 99.98\% \). The number of pixels classified correctly as not being tag centers (true negative, \( tn \)) is not that relevant. The goal is to maximize the number correctly classified tag centers (true positives, \( tp \)) while reducing as much as possible the number of tag centers that were not found (false negatives, \( fn \)) and the pixels that were incorrectly labeled as being tag centers (false positives, \( fp \)). The more robust F1 measure or F1 score [30] was chosen to evaluate the relevant accuracy of the methods explored. It is defined as the harmonic mean between precision and recall:

\[
F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}
\]  

(4.2)

where precision and recall are defined as:

\[
\text{precision} = \frac{tp}{tp + fp}
\]  

(4.3)

\[
\text{recall} = \frac{tp}{tp + fn}
\]

It is important to note that, while F1 score is a better measure than accuracy on skewed datasets, both accuracy and F1 score behave similarly in asymptotic analyses. If a classifier can achieve an accuracy arbitrarily close to 100%, and the dataset contains at least one positive and one negative samples, then it can be trivially proven than the F1 score will also be arbitrarily close to 1.

46
4.2 LoG pipeline

The bee tags were designed to be detected by template matching with a Laplacian of Gaussian (LoG) filter [31] of $\sigma = 2.0$. However, the method formulation in [31] was too ambiguous to be reproduced. A very high accuracy, 89%, is mentioned though neither the dataset nor the definition of accuracy are given. In this section, a different LoG filter based method is enunciated and measurements are listed. Design decisions were based on computational efficiency, training dataset accuracy and common sense.

First of all, the similarity between the tag center and the LoG filter of $\sigma = 2.0$ is obvious, at it can be inferred from figure 4.2.

![Figure 4.2: Typical tag intensity values compared to the LoG template](image)

The hive cells that fill the background have sharp edges and intensities values of the corners of some hexagonal cells do approach the LoG filter in shape. The best background estimate found in the previous section is the median of 1 hour of video. The background is removed by pixel wise subtracting it from the frame image. The resulting image is uneven, with areas previously in shadow showing lower contrast. A simple yet robust lighting correction has to be employed to relieve situations like in figure 4.1(e). One of the simplest is the pixel-wise multiplication of this background removed image with a lighting correction image $(\ell_{x,y})$ that should be constant throughout the 1 hour sequence.

A lighting intensity estimation image $(\hat{l}_{x,y})$ was created from the background image by heavy blurring with a Gaussian filter [21]. A $\sigma$ of 20 pixels was found to work well. The lighting estimation was normalized to the interval $[0, 1]$. The lighting correction image should have high values for low lighting estimates and be close to 1.0 when lighting is sufficient.

Once the image is lighting corrected, the tag centers are detected as the local maxima over a LoG filtered image that have values over a certain threshold. LoG has a symmetrical kernel so template matching has the same result as convolution. The local maxima were points where the LoG filtered image had the same intensity as the image obtained by maximum filtering, using a $11 \times 11$ pixel square structuring element, the LoG filtered image. The typical width of a tag in the video is $15 \times 15$ pixels so no two tag centers should be closer (with respect to the Manhattan distance) than 11 pixels.

It was not obvious how to compute the lighting correction from the lighting estimation and three methods were explored:

1. Linear interpolation: $\ell_{x,y} = 2.0 - \hat{l}_{x,y}$
2. Inverse interpolation: \( lc_{x,y} = \frac{1.0}{l_{x,y}} \)

3. Fillet interpolation: \( lc_{x,y} = \frac{2.0}{l_{x,y} + 1} \)

As it can be also seen in figure 4.3, all methods do not alter the image when the lighting is at maximum (1). Linear and fillet can amplify the contrast my a maximum factor of 2 while the inverse is potentially unbounded. The term fillet is used in that it is a convex rounding of the linear interpolation.

![Comparing the lighting compensation functions](image)

Figure 4.3: Typical frame after background removal and lighting compensation. Candidates detected by the pipeline are marked with small white circles

Each method was evaluated on the 3-frame training set mentioned in chapter 2. The confusion matrix \((tp, fp, fn, tn)\) was computed in the following way. The tag centers predicted by the pipeline were matched with the nearest neighbor from the ground truth tag centers. If the distance to the nearest neighbor was \( \leq 3.0 \), then that predicted point was considered a true positive, otherwise it was considered a false positive. Ground truth tag centers that remained without a match were labeled as false negatives. If a ground truth tag matched two or more predictions, the prediction method was considered invalid. The LoG filter was designed so this would not happen. The number of true negative were calculated with \( 1603072 - (tp + fp + fn) \) since every frame has \( 1616 \cdot 992 = 1603072 \) pixels. The number of true negative is at least two orders of magnitude greater than all other confusion matrix values combined and its exact calculation is of little importance.

The F1 score of each method depends on threshold, each having a different range. To exclude the choice of threshold from the method benchmarking, Receiver operating characteristic (ROC) curves were used [32]. The true positive rate (equal to recall and hit rate) was plotted against the false positive rate (also known as false alarm rate). The false positive rate is given by:

\[
FPR = \frac{fp}{fp + tn}
\]  

(4.4)

and does rely on the true negative count though small variations in \( tn \) do not affect it. Both true positive and false positive rates are multiplicative in classification cascades, which is why they are relevant coordinates in discriminating classifier performance.

The ROC curves for the LoG based method are shown in figure 4.4 for each lighting correction interpolation method.
Figure 4.4: LoG pipeline ROC curves for each lighting compensation function

Unlike generic machine learning methods, where the true positive rate can be increase asymptotically to 1 by raising the false positive rate, LoG filter can only attain a true positive rate of 0.901. Of the three methods employed, inverse interpolation is the farthest from the top left point, meaning it has the least discrimination power. Linear and fillet produce almost identical ROC curves, with fillet performing slightly better.

In terms of F1 score, fillet gives the highest value of 0.735 as listed in table 4.1. Given that fillet shows superior average and best case behavior, it was selected as the lighting correction stage prior to the LoG filter.

Table 4.1: Best results of each interpolation method

<table>
<thead>
<tr>
<th>Method</th>
<th>Expression</th>
<th>Maximum F1 Score</th>
<th>Maximum F1 Score Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear interpolation $l_{x,y} = 2.0 - \hat{i}_{x,y}$</td>
<td>0.730</td>
<td>17.60</td>
<td></td>
</tr>
<tr>
<td>Inverse interpolation $l_{x,y} = \frac{1.0}{\hat{i}_{x,y}}$</td>
<td>0.665</td>
<td>37.50</td>
<td></td>
</tr>
<tr>
<td>Fillet interpolation $l_{x,y} = \frac{2.0}{\hat{i}_{x,y}+1}$</td>
<td>0.735</td>
<td>15.85</td>
<td></td>
</tr>
</tbody>
</table>

The effect the LoG filter threshold has on the accuracy metrics is shown in figure 4.5. The F1 score forms a bell curve around the maximum value. Close to this point both the precision and recall have around the same value. The recall does not decrease considerably from its maximum at threshold 0 up until around threshold 10. Recall or true positive rate is important if the LoG method
is to be used as a preprocessing stage in a cascade of classifiers. If a genuine tag center is rejected at this stage, it cannot be detected in subsequent stages. The LoG pipeline can produce a high number of false positives since they can be filtered later on. Hence, when the pipeline is part of a cascade, a low threshold (less than 10) ought to be used to allow the later classifiers to compensate for its weaknesses.

Based on the previous observations, the standalone LoG pipeline algorithm can be stated as follows:

1. Perform a pass through 1 hour of video and accumulate the temporal histogram.
2. Compute the median from the histogram and use it as a background estimate $bg$.
3. Compute the lighting correction $lc$ as
   \[ lc = \frac{2}{\hat{l} + 1}, \quad \text{where } \hat{l} = \frac{bg \otimes G_{\sigma=20}}{255} \]
   \hspace{1cm} (4.5)
4. For the desired frame $i$ in the 1 hour sequence do:
   
   (a) Subtract the background (median) estimate from the raw frame image and multiply the result the by the lighting correction to obtain the corrected frame $ic = (i - bg) \cdot lc$.
   
   (b) Apply a LoG filter with $\sigma = 2$ to get the tag center likelihood image $ip = ic \otimes LoG_{\sigma=2}$.
   
   (c) Apply a maximum filter with a square $11 \times 11$ flat structuring element to $ic$ yielding $im$.

Figure 4.5: LoG pipeline accuracy metrics versus LoG Threshold when the fillet lighting correction is employed
The tag centers are obtained as the set \( \{ (x, y) | \text{ic}_{x,y} = \text{im}_{x,y} \ \text{and} \ \text{ic}_{x,y} > t \} \), where \( t = 15.85 \) is the threshold.

The detected tag centers on top of a corrected frame are shown in figure 4.6.

Figure 4.6: 20000th frame after background removal and lighting compensation. Candidates detected by the pipeline are marked with small white circles.

Most tags are correctly detected. False positives include reflections off the sugar solutions, bee wings and bodies. Some bees have their rectangular tags glued on top of the old circular tags. Sometimes, the tags underneath become visible and contrast with their surrounding strongly enough to be picked up as false positives. Especially in crowded regions, many tags remain undetected.

The LoG pipeline was tested on three separate test sets. Each test set was composed of a single frame, far apart in display order from the frames used to deduce the pipeline parameters. The confusion matrices, for each dataset are listed in table 4.2. The generalization performance of the pipeline is rather poor. F1 scores are far below the 0.735 training set score. The likely cause is the high threshold, which is the result of overfitting. Using more training frames may give a better, and probably lower, threshold estimate.

<table>
<thead>
<tr>
<th>Test dataset</th>
<th>tp</th>
<th>fp</th>
<th>fn</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>test-hq</td>
<td>46</td>
<td>7</td>
<td>101</td>
<td>0.868</td>
<td>0.313</td>
<td>0.460</td>
</tr>
<tr>
<td>test-mq</td>
<td>42</td>
<td>13</td>
<td>95</td>
<td>0.764</td>
<td>0.307</td>
<td>0.437</td>
</tr>
<tr>
<td>test-lq</td>
<td>36</td>
<td>9</td>
<td>111</td>
<td>0.800</td>
<td>0.245</td>
<td>0.375</td>
</tr>
<tr>
<td>Combined</td>
<td>124</td>
<td>29</td>
<td>307</td>
<td>0.810</td>
<td>0.288</td>
<td>0.425</td>
</tr>
</tbody>
</table>
4.3 Viola-Jones object detection framework

The LoG pipeline was designed by incorporating a great deal of prior knowledge on the design and layout of the tags. The only parameter that had to be learned was a threshold.

In this work, an alternative to hand-designed features is proposed. A pattern recognition system could be provided with a set of positive and negative examples and learn automatically to find tags in unseen images.

One of the fastest image pattern recognition systems available is the Viola-Jones boosted cascade object detection framework [33]. The boosted cascade is a state-of-the-art detector [24] that has been successfully applied to detecting faces and a variety of rigid objects. What makes it different from other pattern recognition methods is its low computational cost and ability to detect objects of varying size in real-time on modest hardware by year 2013 standards [34].

The Viola-Jones method draws its strengths from an innovative combination of three concepts. First, it relies of the extremely computationally efficient Haar-like features. A small subset of these is selected using a modified version of the Adaboost meta-algorithm to obtain a highly accurate binary classifier. Object detection in images requires a rejection rate of the order of a million to one, which would require a classifier employing a large number of features. This monolithic classifier is instead split in a sequence of simpler classifiers called a cascade that has roughly the same accuracy though greatly reduced expected computational cost.

4.3.1 Haar-like Features

For a given candidate pixel, a rectangular area centered at that pixel forms the detection window. A Haar-like feature is computed as the weighted sum over an area inside the detection window. The center of the area does not have to correspond to the center of the window (candidate pixel). The area is made up of a collection of identical adjacent rectangles. The weights are shared across each rectangular regions. Dark rectangles indicate that the weights are negative while white rectangles contain positive weights. The smallest absolute value of a weight is generally set to 1. Hence, all other weights take integer values. The features can be classified by the number of rectangles that comprise them as shown in figure 4.7. For example, the feature value of the leftmost two-rectangle feature is simply the arithmetic difference between the sum of intensities of the pixels that are covered by the right (white) rectangular area and the of intensities of the pixels that fall inside the left (black) area.

![Two-rectangle, three-rectangle, four-rectangle features](image)

Figure 4.7: Haar-like feature prototypes used in the Viola-Jones method

For three-rectangle features, the weighting has to ensure that a completely
uniform window gives a zero feature value. The weights for the middle feature are shown in figure 4.8. The white regions combined have half the area of the black regions and hence double the weight. If the window were subsampled in each dimension by the rectangle sides, the Haar-like features would compute the differential between two subsampled regions. In this sense, the features are effective detectors of edges of their scale. They can also viewed as an approximation of discretized Gabor wavelets [35], where the spatial response values are rounded to the nearest integers. Thus, Haar-like features function in roughly the same way as neurons in the visual cortex of some mammals.

![Figure 4.8: The profile of a three-rectangle feature. For uniform images, the sum of the weights must always be zero.](image)

A simple binary classifier \( h(x) \in \{-1, 1\} \), computed over the image window \((x)\) is constructed from a single feature (feature value \( f(x) \)), a threshold \((th)\) and a polarity \((p \in \{-1, 1\})\). It is computed as:

\[
h(x) = \begin{cases} 
1 & \text{if } f(x) \cdot p \leq th \\
-1 & \text{if } f(x) \cdot p > th
\end{cases}
\]  

(4.6)

By inverting the polarity, the classifiers can be thought of employing one of the symmetric features in figure 4.9.

![Figure 4.9: Symmetric prototypes obtained by inverting the polarity of the original classifiers](image)

The feature set originally proposed in [33] handle diagonal edges using the four-rectangle checkerboard layout. An extended Haar-like feature set approximation was proposed in [36] by adding 45° rotated features as shown in figure 4.10. The checkerboard feature became superfluous in this context and was removed. The center-surround features were introduced, which have a rotational invariance that was missing in the original set. The study quotes 10% reduction in false positives in face detection applications. The greatly increased training time required to support the extended set was considered too great compared to the likely gains as so only the original set was used.
The constraint that the weights be the same across a rectangular region makes possible the evaluation of Haar-like features in constant time, regardless of the size of rectangular area, by using an auxiliary data structure: the integral image [33].

**Integral Image**

The Integral Image value at point \((x, y)\), \((I_{x,y})\) is defined as the sum of the pixel intensities in the original image \((i_{x,y})\) above and to the left of the current location, inclusive:

\[ I_{x,y} = \sum_{x' \leq x} \sum_{y' \leq y} i_{x', y'} \quad (4.7) \]

The intuition behind how the Integral Image simplifies computation is shown in figure 4.11(b). The sum over the pixel values inside a rectangle with the upper left corner coordinates \((x_1, y_1)\) and lower right corner coordinates \((x_2, y_2)\) is given by:

\[ S_{x_1, y_1, x_2, y_2} = \sum_{x_1 \leq x' \leq x_2} \sum_{y_1 \leq y' \leq y_2} \sum_{i_{x,y}} = I_{x_2, y_2} + I_{x_1, y_1} - I_{x_2, y_1} - I_{x_1, y_2} \quad (4.8) \]

The sum requires thus only 4 look-ups. Line features and all other higher order features can reuse some of the look-ups, further increasing the effectiveness of the Integral Image representation. The rationale behind the integral image stems from elementary calculus. The area under a curve over the interval \([a, b]\) of a continuous function \(f : \mathbb{R} \to \mathbb{R}\) is given by \(\int_{a}^{b} f(t) \, dt\). The integral \(F(x) = \int_{a}^{x} f(x') \, dx'\) can be precomputed and thus the area under the curve for any segment \([a, b]\) can be expressed in two look-ups: \(\int_{a}^{b} f(x') \, dx' = F(b) - F(a)\).
The integral image corresponds to the case where the function is the intensity value of the pixel at the given coordinate. The principle can be also applied to any continuous function of a single pixel intensity. In particular, the integral image of the square intensities is also kept, in order to quickly compute the variance of any window. Also, the integration can be performed over a rotated coordinate system. The rotated features can be computed using such a version of the integral image.

The integral image can be constructed in either one pass by a single execution thread, or in two passes in highly parallel fashion. In the first pass, independently, for every row $y$, the cumulative row sum $s_y(x)$ value for every position $x$ can be computed as:

$$s_y(x) = \begin{cases} i_{0,y} & \text{if } x = 0 \\ s_y(x-1) + i_{x,y} & \text{otherwise} \end{cases}$$

The second pass is the calculation of the integral image from the cumulative sums, independently, over every column:

$$I_{x,y} = \begin{cases} s_{x,0} & \text{if } y = 0 \\ s_{y-1}(x) + i_{x,y} & \text{otherwise} \end{cases}$$

Both operations require only one addition per pixel each and highly data parallel, which makes integral image construction overhead comparable to the acquisition of the source image itself.

The integral image allow for rapid evaluation of any binary classifier by means of a small and constant number of look-ups. Despite this advantage, every classifier has poor discrimination ability on its own. The Adaboost meta-algorithm however can combine several weak one feature classifiers into a highly accurate hybrid classifier.

**Detection window normalization**

In the beginning of this section it was shown that normalizing the detection window can substantially increase accuracy (figure 4.1(c)). The LoG pipeline addressed this using local maxima and a low threshold, which is limited in
efficacy. The integral image representation allows for adjusting the Haar-like feature values as if they were obtained from a normalized window.

Gaussian normalization, mentioned in subsection 3.2.1 is both robust to noise and easy to compute using two integral images, an intensity image $I$ and a square intensity image $I'$:

$$I_{x,y} = \sum_{x' \leq x, y' \leq y} i_{x',y'}$$
$$I'_{x,y} = \sum_{x' \leq x, y' \leq y} i_{x',y'}^2$$

Let the detection window have top left corner coordinates $(x, y)$, width $w$, height $h$ and number of pixels $n = w \cdot h$. The window intensity sum $S$ and square intensity sum $S'$ are given by:

$$S = I_{x+w-1,y+h-1} + I_{x-1,y-1} - I_{x+w-1,y-1} - I_{x-1,y+h-1}$$
$$S' = I'_{x+w-1,y+h-1} + I'_{x-1,y-1} - I'_{x+w-1,y-1} - I'_{x-1,y+h-1}$$

The intensity variance over the window can be computed from the integral images as:

$$\sigma = \sqrt{\frac{S'}{n} - \frac{S^2}{n^2}}$$

All Haar-like features were designed to have a value of zero over a uniform detection image. There is no need to subtract the mean in the normalization stage since adding a constant to all pixel values (lighting adjustment) does not affect the feature values. The expression binary classifier over a normalized window becomes:

$$h(x) = \begin{cases} 
1 & \text{if } \frac{f(x)}{\sigma} \cdot p \leq th \\
-1 & \text{if } \frac{f(x)}{\sigma} \cdot p > th 
\end{cases}$$

4.3.2 Adaboost

For a given window size, the number of possible Haar-like features exceed by far the number of pixels in the window. Some of the first pattern recognition methods devised employed a perceptron [37] [22] that discriminated a weighted sum of features values based on a threshold. Most adaptive classifiers are still based on the concept of making a decision after evaluating a predefined set of features for every data sample. The accuracy of such system is highly dependent on the choice of features, which is up to the experimenter.

If all possible Haar-like features were to be evaluated for every sample, even with the integral image representation, the computational cost would be prohibitive. Experimental results mentioned in [33] infer that a subset of the features are sufficient in building an accurate detector. Once the features are selected, an effective way of combining them has to be found.

The Adaboost [38] algorithm provides a computationally tractable solution to both problems. It takes as input the number of weak classifiers the final combined classifiers should have ($T$), the set of weak classifiers and the training data ($X \times \{-1,1\}$). $X$ can take any form as long as every training data sample ($x_i \in X$) is a valid input to any weak classifier in the set. Every training data sample has its true label $y_i \in \{-1,1\}$ associated with it. At every round $t$,
a classifier $h_t$ is chosen and its associated weight $\alpha_t$ is calculated. The final classifier $H$ is actually a perceptron that takes as input the weak classifiers, with the constraint that the threshold be zero:

$$H(x) = \begin{cases} 
1 & \text{if } g(x) > 0 \\
-1 & \text{otherwise}
\end{cases}$$

where $g(x) = \sum_{t=1}^{T} \alpha_t \cdot h_t(x)$ (4.15)

It does so based on a number of assumptions. First, it is a batch learning algorithm in that all the training samples are used at the same time and have to available at every step of training. The final threshold is not needed since it is embedded in the weights. For that to be possible, a weak classifier has to be binary and output $-1$ for a negative prediction and $1$ for positive. Most importantly, Adaboost relies on a separate learning system, called the weak learner to make the actual classifier choice at every round of training.

Unlike the perceptron, which always treats all training examples equally and give weights to features only, Adaboost training relies primarily on weighting the samples. Feature weights are stored but not used as such in the training process itself. The weight at round $t$ of sample $x_i$, $w_{t,i}$, determines how much importance it is given to it during the classifier selection, performed by weak learner. The relationship and data flow between Adaboost and Weak Learner is sketched in figure 4.12.

![Figure 4.12: The two main components of Adaboost and how they interact](image)

**Adaboost proper**

The Adaboost algorithm is stated as:

1. Collect the $m$ training samples $x_i$ and their labels $y_i$
2. Initialize the weights of each sample so as to form a uniform probability distribution: $w_{0,i} = \frac{1}{m}$
3. For each training round $t$ starting at $0$ to $T - 1$:

   (a) Feed the weights to weak learner, which selects the classifier $h_t$ among all possible candidate classifiers $h'_t$ that has the lowest weighted error $\epsilon_t$ on the training set. $\epsilon_t$ is expressed in terms of $h'_t$ as:

   $$\epsilon_{h'_t} = \sum_{i=0}^{m-1} err(i), \text{ where } err(i) = \begin{cases} 
0 & \text{if } h'_t(x_i) = y_i \\
-1 \cdot w_{t,i} & \text{if } h'_t(x_i) \neq y_i
\end{cases}$$

Let $\mathcal{M}$ be the set of sample indices $\mathcal{M} = \{0, ..., m-1\}$. The potential classifier $h'_t$ partitions the training set indices in two mutually disjoint
subsets $\mathcal{M} = OK_{h_t'} \cup NG_{h_t'}$ where

$$
OK_{h_t'} = \{ i \in X \mid h_t'(x_i) = y_i \} \quad \text{and} \quad
NG_{h_t'} = \{ i \in X \mid h_t'(x_i) \neq y_i \}
$$

(4.17)

$OK_{h_t'}$ is the set of indices of the training samples that are correctly predicted by $h_t'$ while $NG_{h_t'}$ is the set of those incorrectly predicted. Since all the error terms in $OK_{h_t'}$ are zero, the weighted error of $h_t'$ can be expressed as:

$$
\epsilon_{h_t'} = \sum_{i \in NG_{h_t'}} w_{t,i}
$$

(4.18)

The simplified notation $OK_t = OK_{h_t}$ and $NG_t = NG_{h_t}$ is used for the partition created by $h_t$.

(b) The correctly classified sample weights are decreased by multiplying them with a constant factor $0 < \beta_t < 1$ while the misclassified sample weights are increased by multiplying with $\frac{1}{\beta_t}$:

$$
w'_{t,i} = \begin{cases} 
w_{t,i} \cdot \beta_t & \text{if } i \in OK_t \\
w_{t,i} \cdot \frac{1}{\beta_t} & \text{if } i \in NG_t
\end{cases}
$$

(4.19)

The resulting weights are normalized in order to form a probability distribution over the training set. The normalization term $norm_t$ is given by:

$$
\text{norm}_t = \sum_{i \in \mathcal{M}} w'_{t,i}
$$

(4.20)

and the weights of the next round are obtained using:

$$
w_{t+1,i} = \frac{w'_{t,i}}{\text{norm}_t}
$$

(4.21)

Intuitively, it makes sense that future classifiers should focus more on the samples that could not be predicted correctly by the previous classifiers in order to augment their strength. This also avoid selecting the same classifier.

4. Construct a confidence measure $g$ that is a linear combination of the classifiers chosen at each round: $g(x) = \sum_{t=0}^{T-1} \alpha_t \cdot h_t(x)$

5. The final classifier is simply the sign of the confidence measure: $H(x) = \text{sign}(g(x))$

The $\beta_t$ values have to be chosen as to prevent the same classifier from being selected again. The weighted error $\epsilon'_t$ on the new weights $w_{t+1,i}$ of the classifier $h_t$ must be as good as random guessing [39]:

$$
\epsilon'_t = \sum_{i \in NG_t} w_{t+1,i} = \frac{1}{2}
$$

(4.22)
The normalization term of the previous weights is:

\[ \text{norm}_t = \sum_{i \in \mathcal{M}} w'_{t,i} = \sum_{i \in \text{OK}_t} w_{t,i} \cdot \beta_t + \sum_{i \in \text{NG}_t} w_{t,i} \cdot \frac{1}{\beta_t} \] (4.23)

The previous weights sum up to one, while over \( \text{NG}_t \) they sum up to \( \epsilon_t \) so the normalization term is actually:

\[ \text{norm}_t = (1 - \epsilon_t) \cdot \beta_t + \epsilon_t \cdot \frac{1}{\beta_t} \] (4.24)

\( \epsilon'_t \) can be expressed in terms of the previous weights as:

\[ \epsilon'_t = \sum_{i \in \text{NG}_t} \frac{w_{t,i}}{\text{norm}_t} \cdot \beta_t = \frac{\epsilon_t \cdot \frac{1}{m}}{(1 - \epsilon_t) \cdot \beta_t + \epsilon_t \cdot \frac{1}{\beta_t}} \] (4.25)

By combining equations (4.22) and (4.25) the value of \( \beta_t \) can be determined:

\[ \beta_t = \sqrt{\frac{\epsilon_t}{1 - \epsilon_t}} \] (4.26)

Adaboost specifies that the final classifier weights \( \alpha_t \) are computed as

\[ \alpha_t = -\ln \beta_t \] (4.27)

The validity of this coefficient choice can be shown by proving that, given enough rounds where the chosen weak classifiers are marginally better than random guessing, the training error can be made arbitrarily small [40].

After all \( T \) rounds are completed, each sample \( x_i \) has the weight \( w_{T,i} \). Since both \( y_i \) and \( h_t(x_i) \) can take only the values \(-1\) or \(1\), we have:

\[ y_i \cdot h_t(x_i) = \begin{cases} 1 & \text{if } i \in \text{OK}_t \\ -1 & \text{if } i \in \text{NG}_t \end{cases} \] (4.28)

hence the update rule becomes:

\[ w_{t+1,i} = \frac{w_{t,i} \cdot \beta_t^{y_i \cdot h_t(x_i)}}{\text{norm}_t} \] (4.29)

By unrolling the update rules, the final weights \( w_{T,i} \) are given by:

\[ w_{T,i} = \frac{1}{m} \cdot \prod_{t=0}^{T-1} \beta_t^{y_i \cdot h_t(x_i)} = \prod_{t=0}^{T-1} \beta_t^{y_i \cdot h_t(x_i)} \] (4.30)

The logarithm of the numerator can be expressed in terms of the confidence measure \( g \) as:

\[ \prod_{t=0}^{T-1} \beta_t^{y_i \cdot h_t(x_i)} = \prod_{t=0}^{T-1} e^{\ln(\beta_t) \cdot y_i \cdot h_t(x_i)} = e^{-y_i \cdot \sum_{t=0}^{T-1} (-\ln(\beta_t) \cdot h_t(x_i))} = e^{-y_i \cdot f(x_i)} \] (4.31)
The weights $w_{T,i}$ are normalized to sum up to 1. Combining this with equations (4.30) and (4.31) yields:

$$\frac{1}{m} \cdot \sum_{i \in M} e^{-y_i \cdot g(x_i)} = \prod_{t=0}^{T-1} \text{norm}_t$$  \hspace{1cm} (4.32)

The training set error of the binary classifier $H(x)$:

$$\epsilon_H = \frac{1}{m} \cdot \text{err}_H(i), \text{ where } \text{err}_H(i) = \begin{cases} 0 & \text{ if } H(x_i) = y_i \\ 1 & \text{ if } H(x_i) \neq y_i \end{cases}$$  \hspace{1cm} (4.33)

can be rewritten as:

$$\epsilon_H = \frac{1}{m} \cdot s(y_i \cdot g(x_i))$$  \hspace{1cm} (4.34)

where $s$ is the step function:

$$s(x) = \begin{cases} 0 & \text{ if } x > 0 \\ 1 & \text{ if } x \leq 0 \end{cases}$$  \hspace{1cm} (4.35)

The inequality:

$$s(x) \leq e^{-x}, \forall x \in \mathbb{R}$$  \hspace{1cm} (4.36)

holds since $e^{-x} \geq e^0 = 1 \ \forall x \leq 0$ and $e^{-x} > 0 \ \forall x > 0$. An empirical proof is shown in figure 4.13.

![Tight upper bound on error](chart.png)

Figure 4.13: $e^{-x}$ as an upper bound for the error step function $s(x)$

The training set error $\epsilon_H$ is thus bounded by:

$$\epsilon_H \leq \frac{1}{m} \cdot \sum_{i=0}^{m-1} e^{-y_i \cdot g(x_i)}$$  \hspace{1cm} (4.37)

Since $\beta_t$ can be computed from the weighted classifier error in that round $\epsilon_t$ (equation 4.26) and the normalization term of that round depends only on $\beta_t$ and $\epsilon_t$ (equation 4.24), the norm is given by:

$$\text{norm}_t = 2 \cdot \sqrt{\epsilon_t \cdot (1 - \epsilon_t)}$$  \hspace{1cm} (4.38)

The training set error upper bound is the product of all normalization terms (equation 4.32) which, by applying (4.38) gives:
If the best classifier \( h_t \) has a weighted error \( \epsilon_t > 0.5 \), then weak learner can just return its opposite \(-h_t\) that will have the error strictly less than 0.5. If the best classifier choice is as good as random guessing, which implies that all available classifiers are the same, the error term would be \( \epsilon_t = 0.5 \). This gives a normalization factor \( \text{norm}_t = 1.0 \), which does not increase the upper bound.

Assuming that, for a given margin \( \delta \), there are arbitrarily many \( (N) \) classifiers that give \( \epsilon_t < 0.5 - \delta \), then the training set error is bounded by

\[
\epsilon_H \leq \frac{1}{m} \cdot \sum_{i=0}^{m-1} e^{-y_i \cdot g(x_i)} = \prod_{t=0}^{T-1} \text{norm}_t = \prod_{t=0}^{T-1} 2 \cdot \sqrt{\epsilon_t \cdot (1 - \epsilon_t)}
\]  

(4.39)

\[
\lim_{N \to \infty} (1 - 4 \cdot \delta^2)^\frac{N}{2} = 0 \implies \lim_{N \to \infty} \epsilon_H = 0, \text{ meaning that the training error can be made arbitrarily low. The bound on generalization error can be expressed in terms of the training error bound [39] and can also be arbitrarily low. Thus, Adaboost is effective both as a feature selector and learning method.}

In summary, the Adaboost proper algorithm can be formulated as:

1. Assign an initial weight \( w_{t,i} = \frac{1}{m}, i = \{1, ..., m\} \) to each of the \( m \) training examples.

2. For every \( t = 1, ..., T \):
   (a) Find the classifier that minimizes the \( D_t(i) \) weighted error:
   \[
   h_t = \arg\min_{h_j \in H} (\epsilon_j) \quad \text{where} \quad \epsilon_j = \sum_{i=0}^{m-1} w_{t,i} \text{ (for} \ y_i \neq h_j(x_i) \text{) as long as} \ \epsilon_j < 0.5
   \]
   If no such classifier can be found, terminate the algorithm.
   (b) Set the \( h_t \) voting weight \( \alpha_t = \frac{1}{2} \cdot \log(\frac{1-\epsilon_t}{\epsilon_t}) \) where \( \epsilon_t \) is the arg min error from step 2 (a)
   \[
   w_{t+1,i} = \frac{w_{t,i} e^{-\alpha_t \cdot y_i \cdot h_t(x_i)}}{\text{norm}_t}
   \]  

(4.41)

where \( \text{norm}_t \) normalizes the equation over all data points \( i \)

3. The resulting classifier is a perceptron with each of the \( T \) classifiers as an input and bias weight of zero:
   \[
   H(x) = \text{sign} \left( \sum_{t=0}^{T-1} \alpha_t \cdot h_t(x) \right)
   \]

(4.42)

One of the drawbacks of emphasizing misclassifications is that outliers and noise in the training data will receive unusually large weights in the final rounds of boosting. Classifiers at this stage end up focusing mainly on finding interpretations of noise [41], ignoring relevant patterns in the data that have a
lower weighting. Empirical tests [41] show however that this rarely happens in practice, even after the training set error has reached zero.

A similar concern is that when a large number of classifiers are chosen with respect to the size of the training set, they may end up overfitting the training data. [42] shows that Adaboost can be defined as a margin maximization algorithm similar to Support Vector Machines (discussed in more details in section 4.4.2) where the margin of a training example \((x, y)\) is given by:

\[
\text{margin}_y(x, y) = \frac{y \cdot \sum_{t=0}^{T-1} \alpha_t \cdot h_t(x)}{\sum_{t=0}^{T-1} |\alpha_t|}
\]  

(4.43)

Well after the strong classifier has achieved 100 % accuracy on the training set, it will continue to improve its margin. As a result, the generalization error can improve even after the training error has reached zero.

**Weak Learner**

While Adaboost proper itself is very efficient in computing the weights of the weak classifiers, it relies on weak learner to make the greedy classifier choice. This is an optimization problem in itself and Adaboost does not does specify how the search should be carried out.

Viola and Jones suggest that the search can be done by brute force for the set of Haar-like feature classifiers [33]. A candidate weak classifier \(h'_t\) has three components, a feature evaluation function \(f\), a polarity \(p\) and a threshold \(t\). For a given feature value and polarity, the number of values the weighted error \(\epsilon_{h'_t}\) can take is finite. If the training samples were to be sorted based on the product \(f(x) \cdot p\), then a given choice of threshold would partition the samples as shown in figure 4.14. Any threshold value within the interval \([f(x_i) \cdot p, f(x_{i+1}) \cdot p]\) will give the same \(\epsilon_{h'_t}\) since it will not change the effect of the classifier in any way. The threshold can be chosen to be the middle of the interval and it is uniquely determined by the sorted index of preceding element in sorted order.

Figure 4.14: How the threshold partitions the training set when the samples are sorted by \(f(x) \cdot p\). Positive samples \((y_i = 1)\) are marked with circles while negatives \((y_i = -1)\) are marked with crosses. Green elements constitute \(OK_{h'_t}\) and red elements \(NG_{h'_t}\).

The threshold and polarity can be computed as the same time by sorting the samples only with respect to the feature value. In a single pass of the sorted samples, for every index, the quantities \(p_{h'_t}(i)\), \(p_{h'_t}(i)\), \(n_{h'_t}(i)\) and \(n_{h'_t}(i)\) are computed. \(p_{h'_t}(i)\) is the sum of the weights of the positive samples that occur before \(i\) in sorted order. \(p_{h'_t}(i)\) is the sum of positive to the right while \(n_{h'_t}(i)\)
and \(nr_{h_t}(i)\) are the corresponding weight sums of negative samples. They are given by:

\[
\begin{align*}
pl_{h_t}(i) &= \sum_{j \in OK_{h_t}, j \leq i} w_{t,j} \quad pr_{h_t}(i) = \sum_{j \in OK_{h_t}, j > i} w_{t,j} \\
nl_{h_t}(i) &= \sum_{j \in NG_{h_t}, j \leq i} w_{t,j} \quad nr_{h_t}(i) = \sum_{j \in NG_{h_t}, j > i} w_{t,j}
\end{align*}
\] (4.44)

\(pl_{h_t}(i)\) and \(nl_{h_t}(i)\) can be efficiently computed recursively, while \(pr_{h_t}(i)\) and \(nr_{h_t}(i)\) can be expressed in terms of \(pl_{h_t}(i)\) and \(nl_{h_t}(i)\):

\[
\begin{align*}
pr_{h_t}(i) &= pl_{h_t}(m - 1) - pl_{h_t}(i) \\
nr_{h_t}(i) &= nl_{h_t}(m - 1) - nl_{h_t}(i)
\end{align*}
\] (4.46)

Let \(\mathcal{M}\) be the set of the sorted indices. Two weighted errors are computed:

\[
\begin{align*}
\epsilon^+_{h,t} &= \min_{i \in \mathcal{M}} pl_{h_t}(i) + nr_{h_t}(i) \\
\epsilon^-_{h,t} &= \min_{i \in \mathcal{M}} pr_{h_t}(i) + nl_{h_t}(i)
\end{align*}
\] (4.48)

If \(\epsilon^+_{h,t}\) is the smaller of the two, then the polarity is chosen to be \(p = 1\). Otherwise the polarity is reversed. The classifier error is \(\epsilon^*_{h,t} = \min(\epsilon^+_{h,t}, \epsilon^-_{h,t})\). This restricts the search to occur only over all possible features and element indices. Both the threshold and polarity can be inferred from these. The classifier returned by weak learner is the one with the lowest error:

\[
h_t = \arg \min_{f \in \mathcal{F}, i \in \mathcal{M}} \epsilon^*_{h,t}
\] (4.50)

where \(\mathcal{F}\) is the set of all possible features. The asymptotic complexity of weak learner is, accounting for worst-case sorting performance, \(O(|\mathcal{F}| \cdot |\mathcal{M}| \cdot \log(|\mathcal{M}|))\). Since Adaboost calls weak learner exactly once per boosting round, the overall training complexity is: \(O(T \cdot |\mathcal{F}| \cdot |\mathcal{M}| \cdot \log(|\mathcal{M}|))\). It is assumed that the features can be evaluated in constant time.

### 4.3.3 Boosted cascade

The cascade concept assumes that for any subset of the training data \(W\), classifier can be trained to achieve a true positive rate greater than a pre-defined threshold \(TPR_{target}\) and a false positive rate less than a predefined \(FPR_{target}\). Let the set of all such classifiers be \(\mathcal{C}_W\). A classifier of this kind that is trained on the entire training data \(c_0\), can act as a filter. \(W\) is the set of all available training samples \(x \in X\) along with their labels \(y \in \{-1, 1\}\), \(W = \{(x, y) | x \in X and y \in \{-1, 1\}\}\). Let \(P\) be the set actual positives in \(W\), \(P = \{(x, y) | (x, y) \in W and y = 1\}\) and \(N\) be the set of actual negatives \(N = W \setminus P\). Let \(W_0\) the set of all elements in \(W\) that are marked positive by \(c_0\), \(W_0 = \{(x, y) | (x, y) \in W and c_0(x) = 1\}\), \(P_0\) and \(N_0\) its positive and negative subsets. The confusion matrix on the training set of \(c_0\) is:

\[
\begin{align*}
tp_0 &= |P_0| \\
fp_0 &= |N_0| \\
fn_0 &= |P| - |P_0| \\
tn_0 &= |N| - |N_0|
\end{align*}
\] (4.51)
Considering the constraints on the true positive rate (recall) $tpr_0$ and false positive rate $fpr_0$:

$$tpr_0 = \frac{tp}{tp + fn} \geq TPR_{\text{target}}$$

$$fpr_0 = \frac{fp}{fp + tn} \leq FPR_{\text{target}}$$

(4.53)

the following relations are obtained:

$$|P_0| \geq TPR_{\text{target}} \cdot |P|$$

$$|N_0| \leq FPR_{\text{target}} \cdot |N|$$

(4.54)

$W_0$ can be used to train another classifier, $c_1$ with a true positive rate greater than $TPR_{\text{target}}$ and a false positive rate less than $FPR_{\text{target}}$ on $W_0$, yielding $W_1$ and $P_1$. The process can be repeated yielding as cascade of $n$ classifiers $c_0, ..., c_{n-1}$. Let $W_k$, $P_k$, $N_k$, $tpr_k$ and $fpr_k$ be defined similarly for any stage $k, k \in \{1, ..., n-1\}$. Equation (4.54) holds:

$$|P_k| \geq TPR_{\text{target}} \cdot |P_{k-1}|$$

$$|N_k| \leq FPR_{\text{target}} \cdot |N_{k-1}|$$

(4.55)

Applying these relations recursively give bounds on the true positive rate $tpr$ and the false positive rate $fpr$ of the entire cascade:

$$tpr \geq TPR_{\text{target}}^n$$

$$fpr \leq FPR_{\text{target}}^n$$

(4.56)

Thus, assuming that suitable classifiers can be found, the number of false negatives and false positive of the cascade be made arbitrarily small resulting an arbitrarily high accuracy on the training set.

It has also been proven that Adaboost can train an arbitrarily accurate strong classifier with far more relaxed constraints. Unlike an Adaboost classifier, where all the weak classifiers have to be evaluated for all samples, the cascade can reduce computation time remarkably.

On the training data, if a sample is negative $x \in N$, the likelihood that it will be present after $k$ classification stages is $P(x \in N_k) \leq FPR_{\text{target}}^k$. If a sample is positive, it will at most be evaluated by all the classifiers in the cascade. Hence, the average number of calls $freq(c_k)$ of classifier $c_k$ a training sample is $freq(c_k) \leq \frac{|N|}{|P|} + FPR_{\text{target}}^k$. The cascade is employed in situations where the number of positives is very small compared to the size of the training data. In these cases, the computation cost reduction is exceptional.

For instance, if the $FPR_{\text{target}} = 0.5$ and the number of positives in the training data negligible, the average cost of evaluating a cascade made up of classifiers with identical computational complexity is less than of computing two classifiers for every data sample, irrespective of the number of classifiers employed. Performance can be further tuned by selecting simpler classifiers for the early stages of the cascade with very complex ones at the end.

Experimental results on face data imply that the cascade exhibits the same behavior on unseen data [33] as on the training data making the cascade a valid learning method.

**Cascade training**

A cascade architecture requires classifiers with strict bounds on the true positive and false positive rates. On the other hand, Adaboost produces classifier with arbitrarily high accuracy but does not offer any guarantees on the true positive
and false positive rates. The strong classifier is a weighted combination of weaker classifiers, thresholded with a value of zero. Adaboost can be altered to include an additional parameter: the weighted sum threshold. This threshold can be adjusted to produce a classifier fit for a cascade. Further work needs to be done in order to prove whether the theoretical guarantees on accuracy of Adaboost apply to this modified version.

The cascade is a degenerate form of a decision tree, where all nodes are restricted to having only right children. Due to the similarities, a cascade can be trained in the same way as a decision tree [43].

First the desired cascade $tpr$, $fpr$, the maximum number of stages $n$ and the maximum number of weak classifiers per stage $T$ have to be set. $TPR_{target}$ and $FPR_{target}$ are computed using:

$$TPR_{target} = \sqrt[n]{tpr}, \quad FPR_{target} = \sqrt[n]{fpr}$$

The set $P$ can be specified explicitly, either as rectangular regions in training image or as the pixel intensity values of these rectangular regions. The extremely high number of negative sample required to train a deep cascade, typically at least a few million, requires a different way of defining $N$.Enumerating the rectangular regions of the negative sample can be a daunting task for a human operator. Instead, $N$ consists of a set of all possible windows in a collection of background images that do not contain any positives.

Each stage uses fixed number of positive samples $n_{pos}$ and negative samples $n_{neg}$ for training that have to be specified beforehand. These samples must have been marked positive by all previous stages. While the set of negative samples is usually enough, $n_{pos}$ must allow sufficient positives to train the last stage of the cascade:

$$n_{pos} \leq |P| \cdot tpr \frac{n_{neg}}{n_{pos}}$$

Let $TPR_k$ and $FPR_k$ be positive and false positive rates of the cascade up until stage $k$ inclusive:

$$TPR_k = \prod_{i=0}^{k-1} tpr_i, \quad FPR_k = \prod_{i=0}^{k-1} fpr_i$$

The cascade training algorithm is shown in algorithm 1. Besides the addition of the threshold, Adaboost is trained for a variable number of rounds. The ability of Adaboost to support resuming, by embedding the state in the weights, makes it an ideal candidate for cascade classifier generation.

**Cascade deployment**

In contrast with the long training times, the boosted cascade offers very low detection times. The actual search for positives is carried out in a near brute force manner. The set of all possible windows is subsampled by a factor of $\Delta$, as shown in figure 4.15. The dotted circles represent the centers of all detection windows that fed into the cascade in order to find positives. Due to the fact that Haar-like features are composed of rectangular regions, they give similar values for slightly translated windows. Despite constraining the window centers to a grid of size $\Delta$, it was observed experimentally that for small values ($\Delta = 2$), the true positive rate of the cascade is not greatly affected.
Algorithm 1 Cascade training

for $k = 0$ to $n - 1$ do

$P_k \leftarrow \emptyset$ $\triangleright$ Create training data for stage $k$

while $|P_k| < n_{pos}$ do $\triangleright$ Obtain positives

Select $x_p \in P$

if $c_0(x_p) = 1$ ... $c_{k-1}(x_p) = 1$ then

$P_k \leftarrow P_k \cup \{x_p\}$

end if

end while

$N_k \leftarrow \emptyset$ $\triangleright$ Obtain negatives

$\text{reject\_count}_k \leftarrow 0$

while $|N_k| < n_{neg}$ do

Select a window $x_n \in N$

if $c_0(x_n) = 1$ ... $c_{k-1} = 1$ then

$N_k \leftarrow N_k \cup \{x_n\}$

else

$\text{reject\_count}_k \leftarrow \text{reject\_count}_k + 1$

end if

end while

if $\frac{\text{reject\_count}_k \cdot n_{neg}}{n_{neg} + n_{pos}} < \text{FPR}$ then

Terminate algorithm

end if

$W_k \leftarrow P_k \cup N_k$

$c_k \leftarrow \text{empty classifier}$ $\triangleright$ Add a new stage

Initialize weights $w_{0,i}$ over $W_k$ according to Adaboost

for $t = 0$ to $T - 1$ do

Add weak classifier $h_t$ with weight $\alpha_t$ to $c_k$ according to Adaboost

Adjust weights $w_{0,i}$ according to Adaboost

threshold $\leftarrow 0$

while $\text{tpr}(c_k) < \text{TPR}_{\text{target}}$ do

Decrease threshold

$c_k \leftarrow \text{sign} \left( \sum_{j=0}^{t} \alpha_j \cdot h_j \right)$

end while

if $\text{fpr}(c_k) = fpr_k \leq \text{FPR}_{\text{target}}$ then

Stop adding weak classifiers and break from this loop

end if

end for

end for
The original Viola-Jones algorithm scans the image at multiple resolutions. The cascade is trained with windows at a fixed resolution. Windows need to be resized in order for the detection to be performed. Instead of scaling the windows, the integral image is accessed at the scaled feature points. Figure 4.16 shows the cascade detection process for every window. In the case of the bee video, the honeybees have roughly the same size and resizing is not necessary.

While the translational variability of Haar-like features does not affect the true positive rate, it does produce a high amount of false positives in the vicinity of a true positive. A detection image, comprising of every window center is created. It is less than the size of the original image with white pixels representing positive detections and black negative. Neighboring detections are grouped by dilating the image with a $3 \times 3$ rectangular structuring element. The connected component centers are taken as detections and their coordinates multiplied by $\Delta$ in order to obtain the detection coordinates in the actual image.
Cascade example

The Haar-like features, arranged by the cascade stages, are shown in figure 4.17 for a state-of-the are face detection cascade created by the inventors of the tilted features [36]. The cascade is displayed in a manner similar to text. Features are drawn from left to right as if they were letter while individual stages listed from top to bottom to form paragraphs. Stages are separated by wider white space.

In practice, Haar-like features are found centered throughout the detection window are often elongated in shape. There are also large variations in area with some features evaluating the whole window while other analyze specific areas.

The first classifier has the fewer features. It is trained on the "simple" data that did not undergo any filtering. Subsequent stages are composed of an increasing number of features. The data that later stages are trained on is more "difficult" in that simpler classifiers were unable process it adequately. Differences between Adaboost and the cascade are evident. In the case of the first classifier, its features are independent of each other. This stems from the fact that the weights are adjusted to prevent similar classifiers from being selected in subsequent rounds, a dissimilarity that extends to the entire stage feature set. However, classifiers in different stages are trained as to both preserve positive and eliminate negatives. As a consequence, corresponding features in neighboring classifier are often similar, in order to preserve the positives. Since the faces are symmetric, features within the same stage tend to be forms pairs that are symmetric.

As shown in figure 4.18, the feature count increases almost linearly with stage index. Considering that the classifier utilization decreases exponentially with stage index, most of the computational burden is placed on the first stages.
Figure 4.17: The Haar-like features utilized in a state-of-the-art front face detector. The features are listed from left to right and the classifiers (stages) from top to down. Stages are separated by wider white space than features. The detection window size $24 \times 24$ pixels. The cascade has 25 stages and a total of 2913 features. It was trained by Rainer Lienhart.
Figure 4.18: The feature count of each stage in the frontal face detector cascade
4.3.4 Results

The Viola-Jones object detector was trained on the train-3 dataset using software included in the OpenCV [20] open source package. Positive samples were obtained by cropping $63 \times 63$ windows, centered at the marked tag centers in each of the 3 frames. A total of $|P| = 434$ positive image windows were collected. Four different background estimates were used as source images for negatives: the average, the median, the 10th percentile and the 90th percentile over 1 hour of video.

The target true positive rate was set at 90%. The maximum cascade depth was chosen to be 20. This gives a per-stage minimum $tpr$ of $tpr = \sqrt[20]{0.9} \approx 0.995$. The number of positives each stage would be trained on ($n_{pos}$) was estimated using equation 4.58 as $n_{pos} = \lfloor |P| \frac{20}{1+20} \rfloor = 394$. It was assumed that classifiers at any stage can be constructed with a maximum $fpr = 0.5$. A 20 stage cascade would have thus $FPR_{target} = \left(\frac{1}{2}\right)^{20} \approx 10^{-6}$, or no more than one in a million false positive rate. An overview of the boosted cascade training parameters is listed in table 4.3. The software package employs an optimization through which training samples whose weights are less than a given percentage ($1 - \text{weight trimming rate}$) are ignored by weak learner.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of stages</td>
<td>20</td>
</tr>
<tr>
<td>Minimum Hit Rate ($tpr$)</td>
<td>99.8%</td>
</tr>
<tr>
<td>Maximum False Alarm Rate ($fpr$)</td>
<td>50%</td>
</tr>
<tr>
<td>Boosting Type</td>
<td>Discrete Adaboost</td>
</tr>
<tr>
<td>Number of positive training samples ($n_{pos}$)</td>
<td>394</td>
</tr>
<tr>
<td>Number of negative training samples ($n_{neg}$)</td>
<td>1000</td>
</tr>
<tr>
<td>Weight trimming rate</td>
<td>95%</td>
</tr>
<tr>
<td>Maximum weak classifier count</td>
<td>100</td>
</tr>
</tbody>
</table>

The training took in excess of 72 hours on the Intel Core i7 computing hardware and had to be terminated after completing 18 stages. Setting a higher $fpr$ may have lowered training time. In any case, the high computational cost of the cascade training leaves out the possibility of parameter refinement by k-fold cross-validation on the training set.

The trained cascade had 18 stages and a total of 116 features. Classifiers have evidently fewer features than in the state-of-the-art face detector, pertaining to the lower number of positive training samples. The features, grouped by stages are shown in figure 4.19(a). Alongside, in figure 4.19(b), the features are shown drawn on top of a typical tag image. The first features of the first stages focus on the center white dot in the center of the tag. Others seem to estimate the contrast between the tag and its surroundings or between the bee body and hive frame. The are even features that detect contrast between rectangular patches on the tag itself. It can be inferred from the rotational asymmetry of many features, that more positive sample have to be specified in order to get a more accurate representation.
Despite the apparent overfitting, the Viola-Jones cascade performs markedly better than the LoG pipeline on the single-frame test datasets. The confusion matrices for every set are listed in table 4.4.

Table 4.4: Viola-Jones performance on the test sets

<table>
<thead>
<tr>
<th>Test dataset</th>
<th>tp</th>
<th>fp</th>
<th>fn</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>test-hq</td>
<td>97</td>
<td>38</td>
<td>50</td>
<td>0.719</td>
<td>0.660</td>
<td>0.688</td>
</tr>
<tr>
<td>test-mq</td>
<td>93</td>
<td>64</td>
<td>44</td>
<td>0.592</td>
<td>0.679</td>
<td>0.633</td>
</tr>
<tr>
<td>test-lq</td>
<td>99</td>
<td>40</td>
<td>48</td>
<td>0.733</td>
<td>0.653</td>
<td>0.691</td>
</tr>
<tr>
<td>Combined</td>
<td>289</td>
<td>142</td>
<td>142</td>
<td>0.671</td>
<td>0.671</td>
<td>0.671</td>
</tr>
</tbody>
</table>

The detected tags in a typical frame (frame D) are shown in figure 4.20. The Gaussian normalization enable the cascade to detect tags in shadow. Some liquid splashes are mistakenly identified as they have a texture that is not present in neither the positive nor the negative training samples.
Figure 4.20: Tags detected in frame D by the Viola-Jones algorithm
4.4 Benchmarks

The MNIST dataset is a popular means of evaluating the performance classifiers [44]. Among top performing methods on this dataset that do not require preprocessing and accept raw pixels as features are support vector machines and multilayer perceptrons. Based on this evidence, both methods were chosen as benchmarks. Both methods have been covered extensively in the literature [45] [22], and will be explained only briefly. As black-box methods, they will be evaluated based on their default specifications. Comprehensive study on the potential of these methods is left for future research.

4.4.1 Multilayer perceptrons

Multilayer perceptrons (MLP) are a form of neural networks that have shown a resurgence in popularity as a general purpose pattern recognition system [45]. MLPs perform regression on a multidimensional input vector by mapping it to an output vector of (mostly) different dimensionality. The mapping consists of a series of linear transformations (matrix multiplications), each followed by a sigmoid squashing function. MLPs can be used in classification by rounding their output, often to a binary value. The most popular training algorithm for neural networks is gradient descent back-propagation [46]. It is based on that fact that derivatives the objective function with respect to each weight (element of a matrix), can be computed efficiently by performing the linear operations in reverse order with the derivative of the sigmoid as a squashing function. Resilient propagation (RProp) [47] is an improvement on gradient descent that uses an acceleration term and takes into account only the sign of the gradient.

4.4.2 Support vector machines

The support vector machine (SVM) is a classification technique first introduced in [29]. It bears resemblance a Perceptron [22]. Given a set of points in high dimensional space, each marked with either a positive or negative label, the SVM strives, just like the perceptron, to find a hyperplane that separates the positive sample from the negative samples. The perceptron convergence procedure stops when such a hyperplane was found. The SVM on the other hand, tries to find the hyperplane that has a maximum margin, i.e. that maximizes the smallest distance from any point in the space to it (figure 4.21). The points that have this maximal margin distance, when projected onto the hyperplane form the support vectors, hence the name. Where the perceptron has potentially an infinite number of possible viable solutions, the SVM has only one. The greater the margin, the more robust is SVM in generalization.

When the data points are not linearly separable (i.e. there is no hyperplane that separates them), the perceptron fails to converge. The SVM instead adds a penalty term in its objective function for every misclassification. The objective function combines margins with the penalty terms.

For very high dimensional spaces, a linear decision boundary is often sufficient [45]. There are cases, however, when accurate classification requires a non-linear boundary. The SVM calculates distances based on the definition of the dot product in the space. This definition can be altered algebraically to warp the space into an infinite dimensional space where a linear boundary can
classify the warped point accurately. The dot product is defined formally as a kernel function. The accuracy of an SVM is highly dependent on the choice of kernel function.

Figure 4.21: A Support Vector Machine as a maximum margin classifier [34]

4.4.3 Benchmark training dataset

Unlike the boosted cascade, both MLP and SVM have to analyze every pixel in every candidate window in order to perform the classification. If either were used on all possible windows, the running time would be prohibitively large. Instead a two-stage cascade can be formed, with a LoG pipeline as the first stage. Similar to the boosted cascade training, the LoG pipeline threshold can be lowered to a desired minimum false positive rate and maximum false positive rate.

Figure 4.22 shows the dependency of the LoG pipeline $tpr$ and $fpr$ on the threshold. The output of the LoG pipeline is used to training both the MLP and SVM. A dataset with an excessive number of negatives would have considerable variance that the benchmarks have to learn, reducing their effectiveness. A false positive rate of around 0.1 should be a soft limit. Naturally, it is desirable to have as high threshold as possible, as the number of false positives decreases sharply as the threshold increases while trying to keep the false positive rate at a high level. The $fpr$ drops significantly at a threshold of 6.45. At this value, the recall is still 0.892, which is 99% of the maximum attainable recall 0.901 on this dataset. The false positive rate is 0.092, which is close to the target 0.1.

The LoG pipeline outputs a set of points. The dataset images, as mentioned in the Materials section, are marked with positive tag centers. Every point produced by the LoG pipeline with the lowered threshold that farther then 3.0 pixels than any positive points is marked as a false positive. The benchmark training negatives are obtained by cropping an image patch of size $63 \times 63$ was from the corresponding dataset image for every false negative point. The original frames were chosen as the source of pixel intensity values. The background removal
Figure 4.22: LoG Pipeline threshold choice. The horizontal lines correspond to precision values of 0.75 and 1.25. They mark the range in which adequate threshold values are to be sought.

do not affect regions near the bees and the lighting correction, designed to increase the performance of the LoG filter produces unnatural images. The purpose of the benchmarks is to complement the LoG pipeline and details that are de-emphasized or lost in the LoG pipeline should be used in this step. The window size was chosen as to be large enough for a human observer to discern whether the detection is truly a tag on a living bee or not. The LoG pipeline output is very skewed with a negative to positive ratio of 10:1. The recall, at 0.9 for one stage, is also much smaller than in the case of the boosted cascade. To compensate for these drawbacks, the set of all positive in the 3-frame dataset are used as benchmark dataset training positives.

A sample of 40 such image patches is shown in figure 4.24. The size of the patches was set as to be small enough to exclude unwanted variations and large enough to be able to discern whether the center was that of a tag and whether the tag was on a bee thorax. Each sample was annotated with an identifier of the image it was extracted from as well as the center coordinates.

**Manual verification**

A verification application was created specifically for this dataset. Based on the image identifier and the center coordinates, a crosshair image was generated and presented to the user. Figure 4.24 shows the actual presentation of both a negative and a positive sample to the user. Only one image is seen by the user at a given time. The original gray values were mapped to the green channel. Red crosshairs were added to the image to aid the user in confirming that the tag center was close enough to the center of the image to be marked as positive. Green was chosen to display the image information as it is the color that human visual system identifies most accurately [21]. The annotations are in the red channels so as to not overwrite any information in the image making it easier to distinguish visual features at the crosshair positions.

The user presses the up arrow key for a positive example and the down arrow
Figure 4.23: Samples of the benchmark dataset training images. White outlines denote positive samples.

(a) Positive Training Example  (b) Negative Training Example

Figure 4.24: Images displayed to the user for a negative example. The tool edits an existing dataset so corrections can be made at any point. Many tag centers were difficult to find in the frame images and are likely to pose problems to an automated system (figure 4.25). The tags were designed to be detected perpendicular to the visual axis. Very often tags are rotated out of plane at such a high angle that it is open to interpretation whether it should be detected at all (figure 4.25(a)). Its projected form is very different from the prototype and identification data is unreadable. Many tags are glued on top of another white tag, which can be easily mistaken for the center white dot with a line protruding from it (figure 4.25(b)). In some cases, the white tags are very tall and, when the tag is rotated out of plane at a wide angle, it no longer has the bee body as background (figure 4.25(c)). Often, the whole bee is not visible, and it is difficult to tell if the tag is part of background or mounted on a bee (figure 4.25(d)). Many tags are in very dark regions and a tag image may have multiple of the previously mentioned problems. All these difficult examples were extracted from a single frame (the 10001st).

Despite enhancement and ease of use, many samples were genuinely difficult to discern manually even after careful inspection. Figure 4.26 shows genuine tag detection border cases. Deciding whether the tag was on a live bee or not was the most frequent problem encountered. Analysis of tag motion across several
frames can however alleviate this.

A supervised learning system is designed to mimic human decision making[22] so its accuracy cannot be expected to go above human level. With many border cases difficult to discern even by a human operator occurring with a high frequency (around 1 in every 10 samples), it cannot be expected that an automated system achieve greater than 10% accuracy based on the 3-frame dataset. Higher accuracies can be obtained by deciding a clear policy on border cases and, most importantly, with a better quality video material.

Figure 4.25: Difficult positives for an automated system
(a) Is the tag on bee, background or part of a dead bee body?

(b) Is the bee alive or not?

(c) Tag or reflection off wing?

Figure 4.26: Difficult positives for a human observer
4.5 Results

Data preprocessing

Data preprocessing impacts performance of both MLPs and SVMs and four different normalization methods were attempted alongside the hidden layer count search: no normalization, Minmax normalization, Gaussian normalization and feature-wise Gaussian normalization. Minmax and Gaussian normalization were carried out in the same manner as mentioned in section 3.2.1. Feature-wise Gaussian normalization considers that the training dataset takes the form of matrix of $A = \{a_{s,f}\}$, where every row $s$ represents a data sample and every column $f$ corresponds to a feature. For instance, the $63 \times 63$ window samples each have exactly 3969 features. Neither fully connected MLPs nor SVMs take into account the order of these features although it can be assumed that pixel $i_{x,y}$, where $x, y \in \{0, ..., 62\}$ makes up feature $f = 63 \cdot x + y$. More generally, let the training data have $n$ samples and $m = 3969$ features. The feature wise mean $\mu_f$ and standard deviation $\sigma_f$ are given by:

$$\mu_f = \frac{1}{n} \cdot \sum_{s=0}^{n-1} a_{s,f} \quad \sigma_f = \frac{1}{n} \cdot \sqrt{\sum_{s=0}^{n-1} (a_{s,f} - \mu_f)^2}$$  \hspace{1cm} (4.60)

Feature-wise normalization function $F$ can be defined as:

$$F(a_{s,f}) = \frac{a_{s,f} - \mu_f}{\sigma_f}$$  \hspace{1cm} (4.61)

Features have their means subtracted and the result is divided by the standard deviation. It is insured that the features have 0 mean and 1 standard deviation.

Cross-validation

A learning system should be tested on a validation set [45] in order to ensure that it was properly trained. For the validation set results to be accurate, its data must not be used in the actual training of the system if the testing is to simulate performance on unseen data. If the data is scarce, a validation set may be too small to allow for accurate measurements. In k-fold cross-validation, the training data is divided into $k$ equal-sized chunks. For every chunk, the system is trained on the remainder of the data and tested on that chunk. The predicted labels of every chunk are concatenated and compared against the actual labels. Thus, the same data can be used as both a training and a validation set with reduced negative effects on generalization performance measurements.

In tuning parameters for the benchmarks, 10-fold cross-validation was used. The positive and negative samples are lumped together. To increase cross-validation effectiveness, the samples were randomly permuted.

80
4.5.1 Multilayer perceptrons

A multilayer perceptron was trained using the resilient propagation learning algorithm. Most parameters were chosen according to the original RProp specification [47] and are listed in table 4.5.

Table 4.5: MLP training parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning algorithm</td>
<td>Resilient propagation</td>
</tr>
<tr>
<td>$\eta_+$</td>
<td>1.2</td>
</tr>
<tr>
<td>$\eta_-$</td>
<td>0.5</td>
</tr>
<tr>
<td>$\Delta_0$</td>
<td>0.1</td>
</tr>
<tr>
<td>$\Delta_{max}$</td>
<td>50</td>
</tr>
<tr>
<td>$\Delta_{min}$</td>
<td>$1.2 \cdot 10^{-7}$</td>
</tr>
<tr>
<td>Weight initialization</td>
<td>Random (Nguyen-Widrow [48])</td>
</tr>
<tr>
<td>Sigmoid function</td>
<td>Logistic $\left( \frac{1}{1+e^{-x}} \right)$</td>
</tr>
<tr>
<td>Termination criteria</td>
<td>MSE change between generations less than 0.01</td>
</tr>
</tbody>
</table>

The single biggest performance factor in MLPs however is architecture. For the purpose of this work, only a single, fully connected hidden layer was chosen. The input layer consists of the raw pixel values of the $63 \times 63$ windows outputted by the LoG pipeline with threshold 6.45. The output layer is made up of a single node. Neurons in the hidden layer are logistic while the neuron in the output later is binary. Choosing the number of nodes in the hidden layer is not trivial. The optimal number of hidden nodes were sought by performing a brute force search in the range $\{2, \ldots, 10\}$. For each node count, the neural network was trained from scratch 10 times. Weight initialization is stochastic, which affects network performance. The F1 score for each of the four preprocessing methods and node count are shown in figure 4.27. The score oscillates wildly from one training round to another. The choice of preprocessing method does not seem to affect the F1 score, with weight initialization being the dominant factor. The scores are low, comparable to random guessing. The variation is higher. For comparison, the F1 score of a network with the output node replaced with a fair coin is shown in figure 4.27(e). The coin mean F1 score is comparable to that of the MLP except that the coin variance is much lower. Increasing hidden layer node count from 2 all the way to 10 does not seem to have any effect and search was not carried out any further.

The performance of using a single node in the hidden layer was not measured since the network would degenerate into a perceptron. SVMs with a linear kernel function in the same way once trained, although the margin maximization of the SVM results in much higher generalization performance.

The training dataset has 4267 samples of 3969 features each. A neural network with $n$ nodes in the hidden layer would have at least $3969 \times n$ weights to tune. The dimensionality of the search space far exceeds the number of samples, which makes the networks prone to overtraining. More training sample would have to be collected in order to train an MLP for this task. Samples can be augmented synthetically by applying different distortions like rotation and noise.

81
MLPs were evaluated as a black-box method and no further experimentation was carried out once the performance was found lacking. The LoG + MLP pipeline was excluded as a viable benchmark.

Figure 4.27: MLP 10-fold performance on the training set
4.5.2 Support vector machines

The kernel function used in training the SVM was chosen to be linear. Radial basis function kernels require excessive computation making them unsuitable in a real-time application. Other kernels are not known to offer good performance on generic data [49]. The 10-fold cross-validation confusion matrices for each of the four preprocessing methods are listed in table 4.6. Gaussian normalization gives by far the best results and it was chosen as the preprocessing method in the LoG + SVM pipeline.

Table 4.6: SVM 10-fold cross-validation performance

<table>
<thead>
<tr>
<th>Normalization</th>
<th>tp</th>
<th>fp</th>
<th>fn</th>
<th>tn</th>
<th>precision</th>
<th>recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>170</td>
<td>250</td>
<td>264</td>
<td>3583</td>
<td>0.405</td>
<td>0.392</td>
<td>0.398</td>
</tr>
<tr>
<td>Minmax</td>
<td>279</td>
<td>135</td>
<td>155</td>
<td>3698</td>
<td>0.674</td>
<td>0.643</td>
<td>0.658</td>
</tr>
<tr>
<td>Gauss</td>
<td>317</td>
<td>98</td>
<td>117</td>
<td>3735</td>
<td>0.764</td>
<td>0.730</td>
<td>0.747</td>
</tr>
<tr>
<td>Feature</td>
<td>174</td>
<td>1251</td>
<td>260</td>
<td>2582</td>
<td>0.122</td>
<td>0.401</td>
<td>0.187</td>
</tr>
</tbody>
</table>

Test set accuracy

The LoG + SVM pipeline, where the data is Gaussian normalized and the SVM kernel function linear was tested on each of the three single-frame datasets. The results are listed in table 4.7.

Table 4.7: LoG + SVM pipeline performance on the test sets

<table>
<thead>
<tr>
<th>Test dataset</th>
<th>tp</th>
<th>fp</th>
<th>fn</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>test-hq</td>
<td>74</td>
<td>19</td>
<td>73</td>
<td>0.796</td>
<td>0.503</td>
<td>0.617</td>
</tr>
<tr>
<td>test-mq</td>
<td>68</td>
<td>23</td>
<td>69</td>
<td>0.747</td>
<td>0.496</td>
<td>0.596</td>
</tr>
<tr>
<td>test-lq</td>
<td>65</td>
<td>11</td>
<td>82</td>
<td>0.855</td>
<td>0.442</td>
<td>0.583</td>
</tr>
<tr>
<td>Combined</td>
<td>207</td>
<td>53</td>
<td>224</td>
<td>0.796</td>
<td>0.480</td>
<td>0.599</td>
</tr>
</tbody>
</table>

The two-stage system performs considerably better than the LoG pipeline on its own yet as well as the boosted cascade. The SVM manages to describe the image patches better, since it has access to pixel values in a much larger local neighborhood. A linear kernel means that the SVM can learn a linear digital filter of any shape, including the LoG filter itself. Using more training data or adding distortions could result in better performance. The SVM learns a linear decision boundary separating 4267 data points in a 3969-dimensional space. Unlike MLPs, which rely on stochastic weight initialization, SVM can converge to a maximum margin boundary, predictively, even with fewer data points than dimensions, if necessary. For example, in a 3-dimensional space, a maximum margin plane between two points can be uniquely defined while there is an infinity of planes that can separate the points. SVM seems well suited in handling scarce datasets such as this one.
4.5.3 Effect of compression on performance

In section 3.1.4 it was established that the size of the video files can be reduced by two orders of magnitude when compressed using the x264 open source H.264 encoder. Some of this decrease can be owed to exploiting spatial and temporal statistical redundancies in the data although most savings are attained by discarding information that is deemed irrelevant to the human visual system. For a scalar quantizer below 28, most of the discarded information is very high frequency noise, which would only impede analysis of the video. From 28 onward, lower frequency data is removed, including tag and bee edges. By quantizer 40, visual quality decreases substantially even for a human observer.

The performance of three tag detection systems is analyzed from the effect compression has on detection F1 measure. To this end, the three single-frame dataset were selected, not only to be far apart from the training set in display order but mostly for the way they are compressed. The test-hq frame is always treated as an I-frame, the first frame in a chunk of 250 consecutive frames that is compressed independently of other blocks. Since all frames in this chunk are predicted based on test-hq, when the video is compressed with a scalar quantizer $Q_P^{video}$, test-hq is compressed using intra-coded prediction with $Q_P^{video} − 3$. test-mq is always a P-frame, exactly in the middle of a 250 frame chunk. It is inter-coded based on the preceding P-frame using a quantizer of $Q_P^{video}$. For high quantizers, the farther away a frame is from the I-frame within a chunk, the poorer its quality, defined by MSE. A P-frame in the middle of the chunk should approximate well the average quality across the video. test-hq is the last B-frame in 250 frame chunk. It is inter-predicted from the preceding and succeeding P-frames with a quantizer of $Q_P^{video} + 2$ ans show be among the worst quality frames in the video.

Only the tag position information from each of the three dataset was used. For quantizer value $QP \in \{0, 4, ..., 40\}$, the tag centers were marked on top of the compressed versions of the three original dataset frames. When LoG pipeline based methods were employed, the median over 1 hour of the compressed video was computed and used in the tests.

All systems were trained once, on the uncompressed version of the train-3 dataset. It was expected that the compression would generally result in lower accuracy metrics as it would alter data in a manner not reflected in the training set.

The F1 scores of the original LoG pipeline (threshold 15.85) on the three datasets, as a function of $Q_P^{video}$ are shown in figure 4.28. In the range $[0, 32]$, $Q_P^{video}$ does not significantly affect the F1 score. The difference between the video quantizer and frame quantizer is evident, with the variation on test-hq (I-frame) shifted by almost 4 compared to the other datasets. For a $Q_P^{video}$ of 24 (28 in the case of test-hq), the F1 score slightly increases. In this range, noise frequencies are suppressed, facilitating detection. At this point, the detection quality is almost the same as when no compression is employed yet the video file size has been reduced by around 60 times. With higher values of $Q_P^{video}$, F1 scores decrease steadily for all frames, with a sharp at 40. As mentioned previously, $Q_P^{value} = 40$ results in very low perceived quality, which affects by the pipeline as well.

The LoG + SVM pipeline, with the SVM having learned a large convolution kernel for a limited number of samples, shows more erratic fluctuations with
\( QP_{value} \). The SVM was trained on samples already containing noise and may have adapted to the noise textures. The suppression high frequencies alters the noise signature, which seems to affect the SVM decision slightly. For P and B-frames, the F1 score actually increases up until \( QP_{value} = 28 \), with noise removal having a positive effect on performance. Afterwards, F1 decreases sharply. For the P-frame, \( QP_{value} = 40 \) is not the absolute minimum and a high detection quality is obtained: \( F1 = 0.562 \). The number of true positives (59) is lower than for \( QP_{value} = 36 \) (tp = 61). The increase in F1 is due to reduced number of false positives (fp = 14, down from 23). High compression removes not only tag edge information but also reflections off bee wings and liquid, which normally result in false positives.

Much more stable performance is exhibited by the Viola-Jones system, as shown in figure 4.30. In the \( QP_{value} \) range \([0, 36]\), no significant fall in F1 score is noticed, with scores oscillating slightly around a high value of 0.7. The B-frame
gives better results than the P-frame as it comes ahead in display order. Hence, the boosted cascade seems to give more focus to general appearance rather than local detail. $QP_{value} = 40$ is the first point where the appearance of the tag is significantly altered resulting in the large drop in F1 score.

Figure 4.30: Effect of compression on Viola-Jones performance

Overall, the Viola-Jones system surpasses the other two system for every test frame and every video quantizer tested, as shown in figures 4.31, 4.32 and 4.33. Over the full F1 scale of $[0, 1]$, the compression has little impact on performance of any system tested.

Figure 4.31: Performance on a best-case I-frame
Figure 4.32: Performance on a average-case P-frame

Figure 4.33: Performance on a worst-case B-frame
4.5.4 Average running times

One of the objectives of this work was to devise a methodology of low enough computational complexity as to allow for real-time processing of video data. Since the available materials only included prerecorded video, streaming raw video data had to be simulated by creating a specialized video file format. The original 1 hour video was decompressed and stored as a collection of 8-bit Windows bitmap images, one for every frame. The frame width being a multiple of four, no padding was used in the bitmap images [50]. A custom video file was created, according to the specifications listed in table 4.8. First, the header was written. Then, for every bitmap image, in display order, the pixel data was extracted and written continuously to the video file. The file was produced using an Intel Core processor, meaning that the integers were stored in little-endian format [13].

The file format was designed so that the raw pixel data would be stored contiguously on a regular hard disk drive (HDD). The device used (WDC WD5000A) could transfer data at 300 MB per second, or approximately 187 frames per second, far higher than the required real-time rate of 14.3 fps. Contiguous data does not require seeking when read. Even when it occurs, for instance in the case of a context switch, the average seek time is 8.9 ms, far below the 69.93 ms duration of a frame. When large amounts of memory are read in order, the HDD controller deduces that data is likely to be read in the same manner in the future and preemptively fetches data succeeding the current read block and stores it into the HDD cache. The built-in capacity of 16MB ensures that the frames can be obtained even more promptly and reliably than from an IP camera.

Table 4.8: Video file format used in speed measurements

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Size</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0−3</td>
<td>4</td>
<td>Signed 32 bit integer</td>
<td>Total number of frames (N)</td>
</tr>
<tr>
<td>4−7</td>
<td>4</td>
<td>Signed 32 bit integer</td>
<td>Size in bytes of a frame (S)</td>
</tr>
<tr>
<td>8−11</td>
<td>4</td>
<td>Signed 32 bit integer</td>
<td>Frame width in pixels</td>
</tr>
<tr>
<td>12−15</td>
<td>4</td>
<td>Signed 32 bit integer</td>
<td>Frame height in pixels</td>
</tr>
<tr>
<td>16−19</td>
<td>4</td>
<td>Signed 32 bit integer</td>
<td>Number of bits per pixel</td>
</tr>
<tr>
<td>20−23</td>
<td>4</td>
<td>Signed 32 bit integer</td>
<td>Size of scan line in bytes</td>
</tr>
<tr>
<td>24−27</td>
<td>4</td>
<td>Signed 32 bit integer</td>
<td>Delay in milliseconds between frames</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28−31</td>
<td>Any 4 byte type</td>
</tr>
<tr>
<td>32−</td>
<td>(N \times S)</td>
<td>Unsigned byte</td>
<td>Pixel data grouped by frames</td>
</tr>
</tbody>
</table>

Using the custom video file, the average wall-clock processing time for each promising method found in this studied was measured. The Linux kernel often buffers recently read data from disk into RAM. The 8.0 GB of RAM of the machine can hold up to almost 5000 frames. To ensure accurate measurements, processing was timed over the span of 10000 frames and divided by 10000. When measurements for different methods are conducted in sequence, the data cached by the previous measurement would be purged before it can be used.

The K-means clustering based edge filtering and image inpainting as well
as the LoG + SVM pipeline were manually multi-threaded to ensure that all 8 CPU virtual cores would be used at the same time. Both were run in 16 threads. The Linux kernel automatically manages the load of these threads ensuring full resource utilization with minimal overhead. The other methods were not computationally complex enough to benefit from multi-threading.

The largest performance increase by optimization was observed in the case of K-means based edge filtering. By running K-means clustering only on the edges detected by the Canny method, an order of magnitude improvement in processing time was obtained. It is the only method that does not run in real time although with careful code vectorization speed should increase to real-time level.

As it can be seen in table 4.9, all three tag detection pipelines can run in real-time. The Viola-Jones method, apart from being the most accurate, also has the lowest running time, even when running on a single thread.

Table 4.9: Average frame processing time (AFPT) over 10000 frames. "1 + 16" denotes that a substantial part of the algorithm runs a single thread with the rest running on 16 threads.

<table>
<thead>
<tr>
<th>Method</th>
<th>Threads</th>
<th>AFPT (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canny edge detector</td>
<td>1</td>
<td>12.47</td>
</tr>
<tr>
<td>K-means frame inpainting</td>
<td>16</td>
<td>1649.15</td>
</tr>
<tr>
<td>K-means edge filtering</td>
<td>16</td>
<td>132.96</td>
</tr>
<tr>
<td>Frame differencing</td>
<td>1</td>
<td>1.23</td>
</tr>
<tr>
<td>EWMA</td>
<td>1</td>
<td>5.62</td>
</tr>
<tr>
<td>Mixture of Gaussians</td>
<td>16</td>
<td>11.23</td>
</tr>
<tr>
<td>Single-pass LoG pipeline</td>
<td>1</td>
<td>37.57</td>
</tr>
<tr>
<td>Single-pass LoG + SVM pipeline</td>
<td>1 + 16</td>
<td>38.10</td>
</tr>
<tr>
<td>(Linear kernel, Gaussian normalization)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single pass LoG + SVM pipeline</td>
<td>1 + 16</td>
<td>1055.25</td>
</tr>
<tr>
<td>(Gaussian kernel, Min-max normalization)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Histogram (median) Creation</td>
<td>16</td>
<td>16.39</td>
</tr>
<tr>
<td>Two-pass LoG pipeline</td>
<td>1 + 16</td>
<td>53.96</td>
</tr>
<tr>
<td>Two-pass LoG + SVM pipeline</td>
<td>1 + 16</td>
<td>54.49</td>
</tr>
<tr>
<td>(Linear kernel, Gaussian normalization)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Viola-Jones detector</td>
<td>1</td>
<td>40.01</td>
</tr>
<tr>
<td>Real-time limit</td>
<td>N/A</td>
<td>69.93</td>
</tr>
</tbody>
</table>
Chapter 5

Discussion and future work

5.1 Background removal

K-means texture cluster analysis was found ineffective as a preprocessing step for tag detection but it shows promise in filtering foreground edge in order to be fed into an edge-based antennae detector. If used only for edge filtering, K-means has a close to real-time performance as initial experiments have shown. More work needs to be done on establishing whether K-means is accurate enough to allow for antenna and trophallaxis detection and how an edge-based detector should be designed.

Frame differencing worked well at preserving fast moving antennae. A great deal of post-processing is required and the exact methodology is left as an open avenue for research. Methods like RANSAC or an adaptive classifier with oriented edge features may be viable options.

The median image over 1 hour of video was found to be the best background estimate and it was removed by subtraction. Online techniques would have been more desirable although their reliance on learning rate made them impractical. A low learning rate required a very long "learning" time, wasting large spans of video time while high learning rate resulted in slow moving bees being assimilated into the background estimate. The mixture of Gaussians method did not perform well, partly due to oversimplification of the model to achieve computational efficiency but mostly due to its reliance on variance in discerning background form foreground. Shadows and bees have the same variance and cannot be distinguished by this method. Future work may focus on estimating several background images, such as hive frame and shadows from the temporal histogram, either by regression or more complex machine learning models.

5.2 Tag detection

Three methods were found to have an adequate tag detection accuracy: a Laplacian of Gaussian based pipeline, a cascade made out of a high true positive rate LoG pipeline and Support Vector Machine and the Viola-Jones object detection framework. The Viola-Jones method had the best performance both in terms of F1 score (0.686) and average computational complexity (40.01 ms per frame on a single thread). The learned cascade had very few features compared to
state-of-the-art face detection systems. Training a boosted cascade with thousands instead of hundreds of positive samples could further increase accuracy. LoG filter based methods have an upper bound of around 0.9 on recall. Many tags do not resemble the "Mexican hat" prototype of the LoG filter and cannot be detected regardless how low a threshold is set. The Viola-Jones method does not have recall limitations due to the flexibility of the Haar-like features. It is possible that, if trained with enough positive samples, the Viola-Jones method could reach near-human performance. Tag labeling does not require any technical knowledge and can be undertaken by researchers with basic understanding of the problem.

A problem shared by all tag detection methods was the inconsistency in detection between consecutive frame or "flickering". Accuracy can be greatly improved by interpolating detection between neighboring frames in display order. The particle filters could be used in generating regions in the frame where bees are likely to move and enforcing detections within that region [51].

The hardware and software barriers pertaining to bee detection in a type A stream have been successfully surmounted. File size was a limiting factor in the amount of video that could be recorded. Using the x264 open source H.264/AVC encoder, the video file size can be reduced by a factor of over 50 with affecting the performance of any of the tag detection systems employed in this work. In particular, the Viola-Jones tag detection system can deliver roughly the same performance as on uncompressed data with videos compressed with a scalar quantizer of 36 that are over 360 times smaller in size.

All background removal and tag detection techniques have been shown to require less than 69 ms of computation per frame, allowing for real time processing. K-means clustering based edge filtering can be made to run in real-time as well by writing code optimized for the Intel Core architecture.

A vital factor in obtaining good performance was the utilization of free or otherwise open-source software technology. In particular, the OpenCV open source computer vision platform has proven to have included implementations of all basic image analysis techniques employed throughout this work and no other platform was utilized in the experiments. Real-time performance and in depth study of the underlying methods was possible by modifying the source code of some OpenCV primitives. The open nature of the platform allows for adequate reproducible of the experiments carried out in this work, making open-source well suited for academic research.

5.3 Future experiments

Even with advanced detection and tracking methods, whether a tag is valid or not is open to interpretation. Manual labeling of positives has proven to be difficult and a high accuracy was not expected from any system. Recent work on ant tracking [52] utilized much higher quality materials. The ants monitored did not rip the tags from their back nor did they walk on top of each other. The background was made out of a uniform synthetic material and no background removal was necessary. Instead of filming at a high framerate with low resolution, framerate was reduced to around 2 fps, resolution was much higher and bursts of light eliminated motion blur and elongated shadows. Most importantly, instead of a custom tag design, the well studied ARTags [53] were
employed. Well tested detection software was freely available. Several lessons can be learned from the success of this work. First, it is apparent that detecting tag images $15 \times 15$ pixels in size is unlikely to be accomplished in a reliable fashion. This work has proven that tag detection is possible at this stage, even with high levels of compression. A trade-off can be made by collecting two video streams at the same time. One stream (A) can retain the current settings. A second stream (B) can have a higher number of pixels, at least $2 \times 2$ or $3 \times 3$ times more and be collected with a lower exposure time, a framerate of 2 fps and under bursts of infrared light. A camera could be programmed to dynamically switch between stream A mode and stream B mode. If this would not be possible, two cameras, one of which can be a programmable still photo camera, can be mounted in close vicinity to have an optical axis as perpendicular to the hive frame as possible. Under these conditions, the frames containing the lighting bursts can be ignored in stream A. The bursts can also help in temporal alignment of the two streams. If large markings on the corners of the hive frame were to be used, possibly large version of the ARTags themselves, the two streams can be accurately spatially registrated. Stream B would allow for ARTags to be mounted on the bees. Instead of mounting tags only on the dorsal part of the thorax, pairs of tags could be mounted either on both the dorsal and ventral sized of the thorax or the abdomen as in [7]. The tags can be detected and identified in stream B. Stream A should only be used in tag detection. Assuming that very fast motion is unlikely to occur in an 0.5 second interval, the information in the two streams can be combined to give an accurate detection and identification of the bees at 14.3 framerate. Lighting should be improved overall by using more infrared LED’s and a light diffuser instead of wide angles, which were found to result in bees casting long shadows.
Chapter 6

Conclusions

Using the x264 encoder, file size has been reduced by two orders of magnitude without loss of relevant detail. This has been confirmed by the fact that compression did not significantly affect the accuracy of any tag detection system tested.

Almost all methodologies analyzed in this work were implemented to run in real-time. Year 2013 computing hardware was not found to be a limiting factor in developing a complete real-time bee interaction tracking system.

Viola-Jones object detection framework proved to be the most suitable for detecting tagged bees. The system delivers high accuracy using only labeled data, without the need of either formal problem description or encoding of prior knowledge. Better results can be obtained with higher quality video and more training samples. The Viola-Jones system is flexible enough to function with other tag designs without the need to write specialized software.

Future filming experiments should employ robust and mature fiducial markers like the ARTags and produce a video stream of higher quality.
Acknowledgements

I would like to thank my supervisor Cris Luengo for allowing me to undertake this degree project in parallel with other courses. Many thanks to Zi Quan Yu for manually tagging the bees and recording the video. My gratitude towards Anders Brun for proposing K-means clustering as a background removal method and for his high expectations in the results of this work. Special thanks to Olle Terenius for enthusiastically explaining many aspects of bee society and for generously providing the computing hardware on which real-time performance was attained. Many thanks to Olle Gallmo for our constructive discussions on machine learning matters. And last, I would like to express esteem towards professor Carlos Adriel Del Carpio at Tohoku University for painstakingly proofreading the manuscript, offering valuable advice on academic writing and especially for his inspirational guidance on scholarly values.
Bibliography


97


Appendix A

Video transcoding

Because of scarcity of free video recording software on the Microsoft Windows platform used by the experimenters, the video has been recorded using Virtual VCR freeware [15]. The only lossless compression method available was the poorly supported Huffman YUV. Transcoding was complicated by the fact that video has been cropped to a width that is not a multiple of 4. As of 2013, this video frame size is no longer supported by libavcodec, which is currently among the most popular free video transcoding libraries. It is employed by both FFmpeg [54] and Mencoder [14] software projects. A fork of FFmpeg, called libav [55], contains a modified variant of libavcodec that does not have this limitation. However, the version libav utilized (9.3) had poor container support for long video files. Mencoder SVN-r31628-4.4.4 or earlier and FFmpeg 0.7.1 or earlier were found suitable. Both are command line applications with extensive manuals.

FFmpeg and, especially, Mencoder were designed to transcode motion pictures and television programming and thus offer extensive support for interlacing/deinterlacing and telecine. Their manuals also focus on encoding video for the best possible viewing quality and in formats that are popular in the entertainment industry. By contrast, the video utilized in this research project requires a constant quality throughout the sequence and was obtained from a digital camera as a series of progressive frames.

Going through the official documentation and extracting relevant command line options has proven tedious both because of the sheer number of options and of the fact that parts of the documentation were out of date. To those who may want to employ video transcoding in their research, table A.1 has been when compiled with ease of use in mind.
<table>
<thead>
<tr>
<th>Option</th>
<th>Mencoder</th>
<th>FFmpeg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input file</td>
<td>[input file]</td>
<td>-i [input file]</td>
</tr>
</tbody>
</table>
| Seek to                      | -ss [hh]:[mm]:[ss]  
|                              | -ss [hh]:[mm]:[ss]  | -ss [hh]:[mm]:[ss]  |
| Duration                     | -endpos [hh]:[mm]:[ss]  
|                              | -t [hh]:[mm]:[ss]     | -t [hh]:[mm]:[ss]     |
| Video filter specifier       | -vf      | -vf |
| Crop                         | -vf crop=[w]:[h]:[x]:[y]  
|                              | -vf "crop=[w]:[h]:[x]:[y]"  | -vf "crop=[w]:[h]:[x]:[y]" |
| Rotate 180                   | -vf mirror=yes,flip=yes  
|                              | -vf "hflip,vflip"     | -vf "hflip,vflip"     |
| Output to grayscale          | -vf format=y8     | -pix_fmt gray |
| Output to H.264 color space  | -vf format=i420   | -pix_fmt yuv420p |
| Remove sound                 | -nosound        | N/A    |
| Raw video output             | -ovc raw       | -vcodec rawvideo |
| Compress video with H.264    | -ovc x264      | -c:v libx264 |
| Lossless H.264               | -ovc x264 x264encopts qp=0  
|                              | -c:v libx264 -qp 0  |
| No-DCT lossless H.264        | N/A          | -c:v libx264 -qp 0 -preset ultrafast |
| Output framerate             | -ofps [frames],[fraction]  
|                              | -vf fps [frames],[fraction]   |
| Output to images             | $ mplayer [Mencoder options] -vo png -frames  
|                              | [name]%08d.png  
|                              | [output file]     |
| Output file                  | -o [output file] | [output file] |
| Get file information         | $ midentify [video file] | $ ffprobe [video file] |

a The values in square brackets are to be replaced with corresponding integer values. When the same symbol is used several times, the number of times the symbol occurs corresponds to the number of digits that must be specified.

b If several filters are used, they should be separated by commas, without any spaces in between. The filters are processed in the order they appear and apply to the output file.

c If several filters are specified, all the options have to be enclosed in double quotes and be separated by commas. The filters will be processed in the order they appear and only affect the file whose name is encountered first after the -vf tag.

d Mencoder outputs images named by the C standard library printf formula "%08d.png" in the working directory where the command was invoked, which may differ from the directory of the source video file. The output path and naming conventions are fixed. Custom path can be obtained by moving and renaming the images after they have been created.
Appendix B

Improving the custom tags through ID choice

In the videos analyzed in this work, the experiments chose the tag IDs in sequence, starting from zero to the number of tags. Many tags got lost either because the bees managed to remove the tags from their back or died and had to be removed along with the tags. Still the IDs did not go beyond 500. By imposing constraints on the IDs, their range can be extended and tags that are easier to be analyzed by an automated system can be produced.

Originally, each of the 8 patches can represent a base 3 digit that is independent from the others. The number of possible IDs encoded this way is \(3^8 = 6561\). Shadows affect both the brightness and contrast of the tags. The center white dot has the highest possible gray value (255) though there is no guaranteed low value. If the tags were constrained to always contain at least one black patch (encoding digit 0), the original tag gray values could be restored through min-max normalization. The number of IDs under these constraints is the number of IDs without constrains minus the number of ID that do not have any black patch at all. A non-black patch can only encode 2 possible values (1 or 2) and these total \(2^8\) combinations. Thus, the number of patches that contain at least a black patch is given by \(3^8 - 2^8 = 6305\). The range reduction is small compared to the potential benefits.

Let \(N_{\text{total}} = 6561\) be the total number of IDs under no constraints. Let \(N_{i_1,i_2,...}\) be the number of IDs that don’t contain digits \(i_1, i_2\) and so forth.

If the center dot and line would be left out of the min-max normalization, it may be useful to always have at least one 0 patch and one 2 patch. The total number of tags under these constraints is 
\[
N_{\text{total}} - N_0 - N_2 + N_{0,2} = 3^8 - 2^8 - 2^8 + 1^8 = 6050.
\]

If gray levels are distorted in a non-linear fashion, it may be necessary to have at least one of each digits in order to restore the tags by analyzing the histogram. The number of IDs would be in this case 
\[
N_{\text{total}} - N_0 - N_1 - N_2 + N_{0,1} + N_{1,2} + N_{2,0} - N_{0,1,2} = 3^8 - 2^8 - 2^8 - 2^8 + 1^8 + 1^8 + 1^8 - 0 = 5796.
\]

Again, even under heavy constraints, the ID range is large enough to easily accommodate a whole hive.

If transitions between neighboring patches were detected, detection could be simplified with neighboring patches on either side would encode different digits.
Patches on one side need not differ from the other as the center dot and line are guaranteed to differ in gray levels. Without affecting the result, it can be assumed that the patches in the front left and front right of the tag (the white line points towards from front) can take any of the 3 values while all others are restricted to have only 2 values. The total number of IDs would be thus \((3 \cdot 2^3) \cdot (3 \cdot 2^3) = 3^2 \cdot 2^6 = 576\). This is still enough to encode a small hive and the limited choice of IDs can be used in error checking. The probability of a decoded ID being valid, assuming uniformly random error would be \(\frac{576}{6561} = 8.78\%\), which would perform better than a 3 bit checksum (12.5%).

For larger ID-ranges, the checksum could be embedded in the ID choice with greater ranges allowing for greater choice.

In general, if \(k\) is the base of the digit being encoded in a tag, \(p\) the number of digits, \(n\) the number of digits that must occur at least once in each tag, then the number of IDs is given by

\[
N = N_{total} - \sum_{i_1=0}^{k} N_{i_1} + \sum_{i_1, i_2=0, i_1 \neq i_2}^{k} N_{i_1, i_2} - \sum_{i_1, i_2, i_3=0, i_1 \neq i_2, i_2 \neq i_3}^{k} N_{i_1, i_2, i_3} + \ldots \tag{B.1}
\]

which can be expressed in terms of combinations \(C^q_n = \frac{n!}{q!(n-q)!}\) as:

\[
N = \sum_{q=0}^{k} (-1)^q \cdot C^q_n \cdot (k - q)^p \tag{B.2}
\]