Politechnika Wrocławska

Faculty of Computer Science and Management
Field of study: **COMPUTER SCIENCE**

Bachelor Thesis

Simple object state detection using camera image

Filip Drapejkowski

keywords:
Image recognition, Computer Vision, Embedded Systems

short summary:
This thesis focuses on the problem of object state detection using camera. Objects are defined as batches of cracked chicken eggs processed by industrial egg cracking machine. A decision is made, whether or not the batch contains damaged egg yolk. If it does, the whole batch is removed from processing line. The system was implemented in embedded environment and tested on exemplary data from RZ-1 industrial machine. System can be used by pharmaceutics companies that produce medicines from egg white, confectionaries and sport nutrition producers.

Supervisor	Dr. Inż. Wocjeich Lorkiewicz	.....	.....
	Title/ degree/ name and surname	*grade*	*signature*

For the purposes of archival thesis qualified to: *
a) Category A (perpetual files)
b) Category BE 50 (subject to expertise after 50 years)
* Delete as appropriate

stamp of the faculty

Wrocław 2015

Streszczenie

W poniższej pracy przedstawiono proces projektowania i tworzenia systemu, który identyfikuje stan obiektów przy użyciu obrazu z kamery. Obiekty stanowią partie rozbitych jaj kurzych, które przypisywane są do jednej z dwóch klas. Czysty produkt jest zdefiniowany jako nieuszkodzone żółtka jaj otoczone białkiem jajecznym albo pusta linia produkcyjna, zaś partie zawierające uszkodzone żółtka albo nieuszkodzone żółtka otoczone szczątkami uszkodzonych stanowią odpad, który usuwany jest z linii fabrycznej.

W treści pracy szczegółowo opisano proces rozbijania jaj, oraz oddzielania białka od żółtka. Następnie przeanalizowano rynkowy popyt na system automatyzujący ten proces. Przeprowadzono badania jakościowe istniejących rozwiązań dla opisanego problemu oraz usprawnień, jakie można w nich dokonać. Zaprojektowano urządzenie, które montowane będzie na maszynach rozbijających jaja firmy OVO-TECH w celu poprawienia ich skuteczności. Omówiono konstrukcję i implementację systemu zawierającego to urządzenie, a także wykorzystane w nim algorytmy. System ten, wykorzystujący platformę komputerową Raspberry Pi i podłączoną do niej kamerę rozpoznaje w widocznej masie jajecznej nienaruszone żółtka, oraz liczy ilość zanieczyszczonego produktu.

Autor poniższej pracy przewiduje, że wykonany system zastosowanie w firmach farmaceutycznych produkujących leki z białka jajecznego, w cukierniach, które potrzebują oddzielania białek od żółtek do procesów piekarniczych takich jak ubijanie piany z białek, oraz w firmach produkujących odżywki dla sportowców.

Abstract

In this thesis, process of creating a device that identifies objects state using camera image is described. Objects are defined as batches of cracked chicken eggs, which are assigned to one of the two classes. Clean product is defined as intact yolks surrounded with egg white or clear processing plant line. Batches containing damaged yolk or intact yolk surrounded with damaged yolk parts are considered waste and removed from processing plant.

Process of egg cracking and separating egg white from egg yolk was described in detail. Than the market demand on a system automatising this process was analysed. A research about quality of existing solutions for such problem was conducted. Possible improvements in them were considered. A device which will be mounted on egg-breaking OVO-TECH company machines was designed. Structure, used algorithms and implementation of the system were described. The system utilises Raspberry Pi embedded platform and connected camera to recognise intact egg yolks in the mass present on the image, and counts the amount of contaminated product.

Author of this thesis is expecting, that developed system will be applied in pharmaceutics companies that produce medicines from egg white, confectionaries and sport nutrition producers.

Contents

1 Introduction

1.1 Industrial egg processing

Chicken egg is a widely used ingredient of food, athletes nutrition and medicines. Simplified structure of chicken egg is commonly known (see figure 1.1.1). Prior to processing or consuming an egg, its shell has to be removed first, what is done in cracking process (see figure 1.1.2). It is estimated that 1.8 trillion chicken eggs are used yearly around the world[1], and most of them are consumed without their shells.

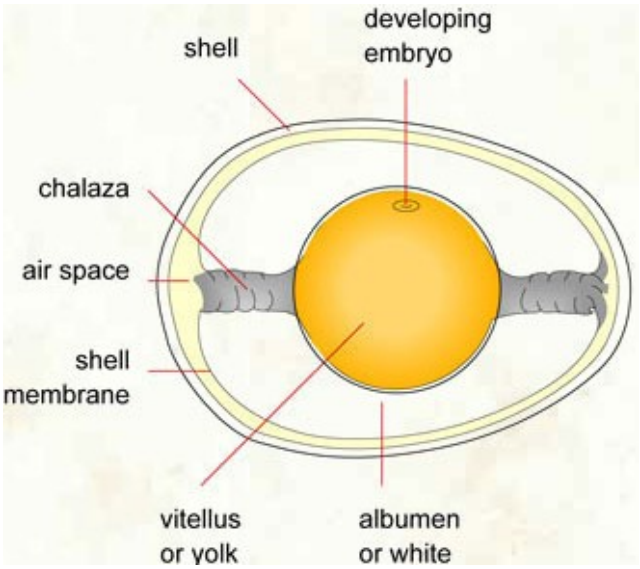

Figure 1.1.1: chicken egg structure, Src: www.gardening-for-wildlife.com

Figure 1.1.2: Manual egg cracking, Src: www.media.eggs.ca

For many companies, raw eggs are required to be not only removed from shells, but also egg-yolk and egg white has to be separated. In the following thesis an industrial problem - large scale chicken egg white and yolk separation - is analysed in terms of possible automation and improvements in currently used techniques. This is due to following applications:

- Confectioneries and bakeries utilize egg white foam as an ingredient for their products. Egg white foam is obtained by aeration (also known as beating / foaming / whipping) process. Comprehensive process description was provided by one of the bakers:
  "When air is incorporated into a liquid or viscous solution, the solution entraps the air bubbles, forming a foam. If the foam is stabilized by proteins, it leavens a food, increasing its height and reducing its density. The viscosity of all egg products is ideal for incorporating air cells during the whipping or beating process. As whipping or beating progresses, air bubbles decrease in size and increase in number, all the time surrounded by egg proteins. Liquid egg products have low air-liquid interfacial tension; thus, when eggs are beaten or whipped, the proteins denature, or simply, they unfold. This exposes two oppositely charged ends of the protein molecule: the hydrophobic, or water-hating end, and the hydrophilic, or water-loving ends. The proteins align themselves between the air and water, securing the air bubbles with their hydrophilic chains pointing into the water and dangling their hydrophobic chains in the air. During baking, these proteins bond with each other, forming a delicate, yet reinforced network.
  Egg whites do this much better than yolks because of the unique proteins found in whites. In fact, even though the term foam technically refers to any system where there are entrapped air bubbles, in the food industry, when discussing egg white products, the term tends to be exclusive to egg whites foams. This is because egg whites, unlike any other natural food ingredient, are able to create the largest possible food foam, a foam six to eight times greater in volume than unwhipped, non-aerated liquid egg white
  Whole eggs and egg yolks can also increase the volume of foods, including certain baked goods and dairy desserts such as ice cream and custard, but just not as much as egg whites alone. Visually, whipped yolks may double or triple in volume, while whipped whole eggs produce less volume than either yolks or whites whipped separately. They are also less thick than yolks alone." [2]
- Biotechnology companies make attempts to manufacture medicines for rare diseases from egg white. This is new and emerging branch of pharmaceutical industry, yet very promissing. For example, Synageva Biopharma that their life-saving proteine from egg whites was recently (February 2015) acquired by Alexion for $8.4 billion.
- Athletes nutrition producers produces their powders and tabs that are supposed to accelerate muscle growth from egg whites, while yolks are often treated as waste
- Dried egg powder that is used to extend egg edibility time is also often product separately from egg white and egg yolk.

All those facts lead to conclusion, that there exist a large market and potential demand for solutions that would crack the egg and separate egg whites from egg yolks in an automatic way.

Author of this thesis managed to find four companies that already produce machines addressing such problem:

- Egg King, located in USA
- Ovobel in Belgium
- Sanovo from Denmark
- Ovo-tech from Poland

Ovo-tech company was contacted by author and enquired about the way the machines they product operate. It appeared, that during last 4 years multiple attempts were made by this company to separate whites from yolks of freshly broken eggs in a mechanic way. Obtained solution (RZ-1 separation module) realise that goal partially, since the separation is imperfect and its ratio varies depending on egg quality. It proved to be working good enough for some companies (see figure 1.1.3), while other rejected it and declared intention to buy OVO-TECH machines in the future, provided that adequate improvements will be made.

Nine companies were inquired whether or not would they invest in improved Ovo-tech machine, if that egg white will be visually clean of yolk parts. Eight of them replied positively that they would seriously consider such offer since their processing plants would benefit in terms of product quality or manpower cost on this improvement.

One of these companies is particularly interesting, since it has purchased some ovo-tech machines, and is nevertheless displeased with its efficiency. This company currently is undergoing a procedure of obtaining US Food Drug Administration (FDA) approval for wide distribution of their egg-based medicine. Executives of this client suspect that using machines with better egg separation will increase their chance for positive outcome of above process.

Figure 1.1.3: Ovo-tech egg braking machines operating over the world

1.2 Problem formulation

Ovo-tech observed that a simple dependence exists: the older the egg is, the more likely its yolk is degenerated inside the egg shell. Also, if an egg processed by Ovo-tech machines is old or the chicken was feed improperly the yolk might might break during the cracking phase (see figure 1.2.1 and 1.2.2). Also, a yolk damage is possible due to cracker imperfections such us being calibrated for different size or weight eggs.

A mixture of damaged yolk and white is created in a result of processing a bad quality egg (old and/or damaged), and it is not possible to properly separate it into two initial components using the RZ-1 separation module. An egg batch is defined as a mixture of freshly broken eggs containing egg yolks, egg white, not containing the egg shells, that is present on a the sliding slope that transports the product to separating module. Eliminating the batches containing fuzzy yolk should decrease chances of accidentally processing the contaminated eggs, while the risk of egg contamination generally increases with egg age.

Ovo-tech deals with those cases by encouraging companies to hire a person that manually removes badly cracked eggs from an accumulating batch (See 2.1 subchapter). Nevertheless, Ovo-tech is dissatisfied about cost that is additionally generated by this solution and the fact that it significantly slows down the processing speed of the entire processing plant.

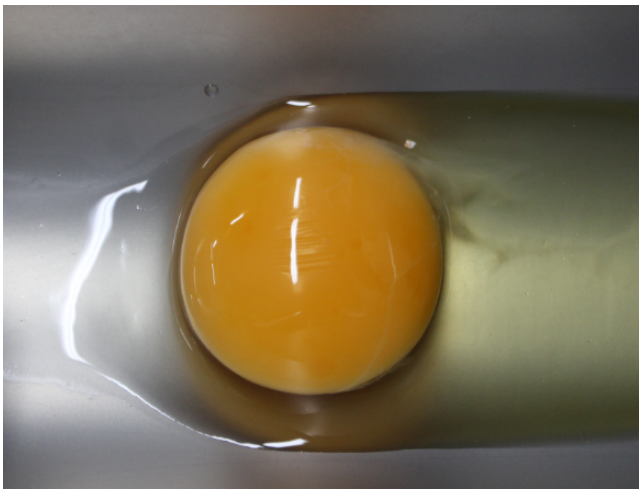

Figure 1.2.1: Properly cracked egg

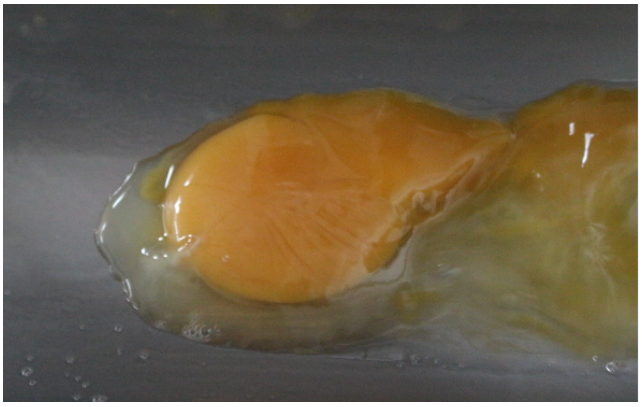

Figure 1.2.2: Damaged yolk in improperly cracked egg

The above analysis led to conclusion, that researching egg-white separation is justified with a significant world-wide market demand. Currently used methods require improvements and an attempt to conduct them is included in this project. Thus, thesis goals are formulated as:

1. Finding an effective method for automatically detecting yolk-and white mixture state.
2. Implementing such method in a prototype device that utilises the discovered method.

2 Problem analysis

2.1 Host machine

Ovo-tech produces a processing plant called "RZ-1" (see figure 2.1.1 and 2.1.2). This device automatically cracks eggs and attempt to separate egg yolk from egg white. This machine has been chosen as the one that will be extended with module prototype prepared in this thesis, due to its availability (it is the most popular of the Ovo-tech machines) and the fact, that it operates relatively slow (compared to other Ovo-tech machines), so the flowing eggs yolks surfaces do not touch each other - what makes them easier to process.

Applications	**Egg breaker used to produce fresh liquid egg. Enable to separate white from yolks.**
How it works	The machine imitates work of people. Egg shells are cut and opened with use of special knives. Hygienic standard of broken eggs is very high.
Destination	Bakeries, confectioneries, egg processing companies.
Capacity	max 3600 eggs/h (separating function off) 1800 eggs/h (separating function on)
Power input	0.37 kW, 1 × 230 V, 50~60 Hz
Dimensions	900 × 1600 × 1100(h) mm
Minimal workspace	2000 × 1500 mm
Weight	ca. 140 kg
Operated by	1–3 people
Delivery time	5 weeks
Note	The machine includes quality control module disallowing weak yolks get into final product. The machine is equipped with yolks-whites separator, and also has 2 modes: fast and precisely.

Figure 2.1.1: Rz-1 unit basic parameters

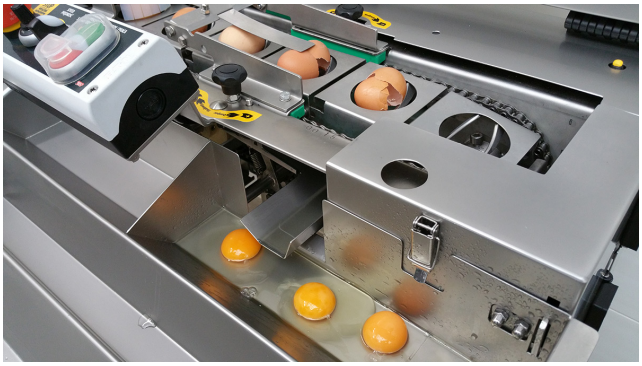

Figure 2.1.2: Egg cracking module of RZ-1 unit

The RZ-1 machine operates in a following way (see figure 2.1.3):

1. Eggs are directed into one-line flow, and mechanically aligned
2. Cracking module cuts the incoming egg surface from below with two knives that are aligned in direction of egg focal radius.
3. The knives imitate human hands work by opening the previously notched egg.
4. The cracked eggs are accumulated in a movable utensil for optical assessment. If the egg yolk appears to be broken, it is removed by an operator from the processing plant.
5. Cracked eggs are transported by sliding to white and yolk separating module.
6. Separation is done in a mechanic way while eggs slide over specificly shaped gap that only egg white fills in, while the egg yolks remain intact.

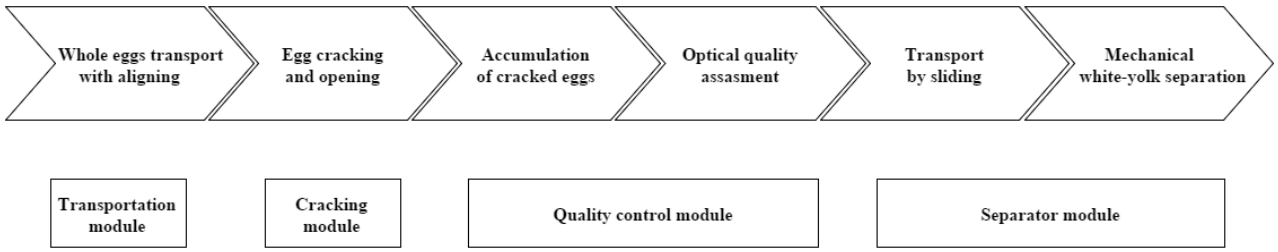

Figure 2.1.3: RZ-1 Egg processing - process and plant structure

The authors idea is to automatise the whole Quality Control module and replace the human expert with electronic processing module, that will automatically asses broken egg quality and send a signal to mechanical part that will remove them mechanically. The creation of the mechanical part of this system is out of the scope of this work and is handled by Ovo-tech engineers. The egg accumulation will not longer be required.

RZ-1 operates in two modes:

- Fast mode
- Precision mode

Fast mode is used if the yolk-white separation is not conducted (separator module is not attached) or if the eggs are very fresh (1-2 days old).
Precision mode is used widely with bad quality eggs to extend amount of time for optical assessment in (5) phase. When Precision mode operates, a few centimeters gap between egg yolks is visible, what highly simplifies the recognition problem. Thus, only Precision mode is considered further to simplify the problem analysis.

Ovo-tech produces also larger scale, more efficient models such us RZ-3, RZ-6 and RZ-8. They also would benefit on assessment automatisation, but are beyond the scope of this work.

2.2 Establishing framework for product cleanness assessment

The requirement is to create a system able to detect all product batches containing badly-cracked eggs. A batch is defined as a mixture of freshly broken eggs containing egg yolks, egg white, not containing the egg shells, that is present on a the sliding slope that transports the product to separating module.

Ovo-tech was unable to specify a short, quantifiable requirement that defines which batch is acceptable and which not. Instead, pictures of batches during machine operation were classified manually by Ovo-tech expert. He decided that batch intact egg yolks should be removed from processing line if (see figure 2.2.1):

1. intact yolk is surrounded by homogeneous mixture of another damaged yolk and white. This is the most strict (in terms of cleanness) case, that generates less strict cases:
2. intact yolk surrounded by damaged yolk blobs
3. fuzzy (damaged) yolk that still keeps its shape without mixing with white
4. damaged egg blobs surrounded with white
5. egg-white homogeneous mixture
6. egg-white non-homogeneous mixture

Two cases are accepted as proper product:

7. intact yolk surrounded with egg white
8. clean egg white also, there is no need to do anything when:
9. transportation part is empty (machine is not operating or the operators are preparing next batch of eggs).

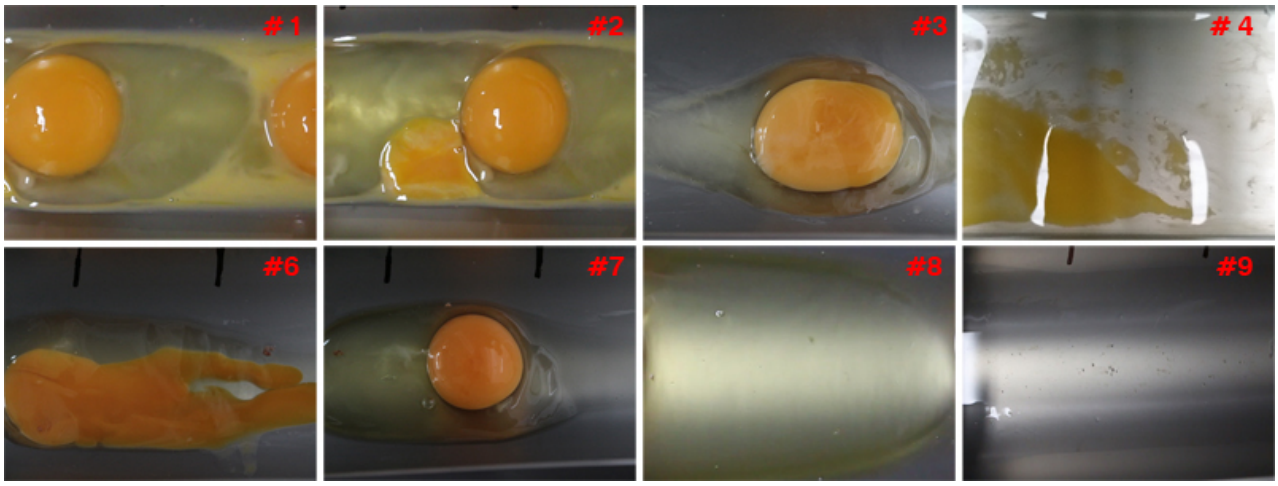

Figure 2.2.1: Cracked egg product cases.

Also, there an additional problem is constituted by chalaza (see figure 2.2.2) - a usually white element that is attached to egg yolk for in-egg suspension. Despite the fact, that client decided that chalaza position and amount are irrelevant to him, its appearance may increase difficulty of the recognition problem if identified as parts of damaged egg yolk.

Figure 2.2.2: Chalaza attached to egg.

2.3 Techniques of automatic egg quality assessment

Different techniques for automatic, non-destructive egg quality grading have been investigated. Among them, the following are worth mentioning:

- Intelligent systems based on visible-infrared transmittance spectroscopy.
  Such systems detect visible infrared (400–1100 nm) light that is altered while passing through the egg with intact shell (see figure 2.3.1). Those alterations are assessed in terms of their dependence on eggs age. Choosing features of this altered egg that are used for detection is a complex problem - authors of this method use Genetic Algorithms and Principal Component Analysis to solve it [3].
- Fourier frequency analysis of a vibration-based response on impact force.
  This method focuses on distinguishing eggs which shell is partially broken from those who are intact by generating some impact force and investigating the acoustic response. The registered responses are assigned broken or intact class using linear Support Vector Machines method, that generates a hypersurface separating these cases.[4]
- Ultrasonic, magnetic resonance, electric resistance, electric nose[5]

All methods listed above require very specific and expensive equipment, are prone to overfitting and take in to account an assumption that does not has to be satisfied for problem considered in this thesis: the egg shell should not be damaged.

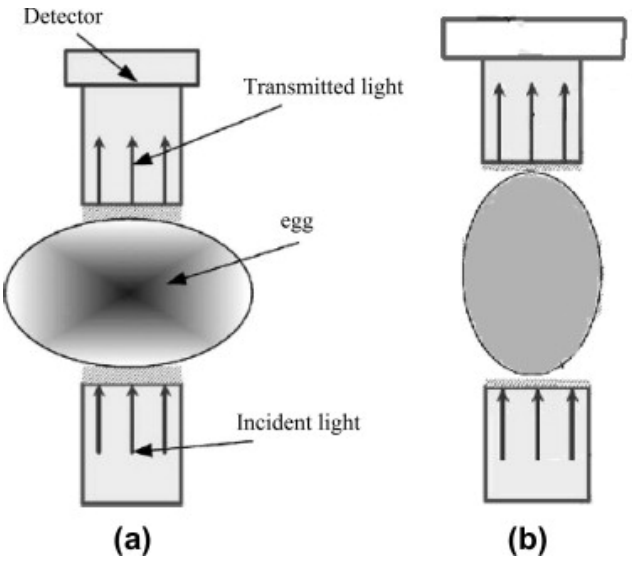

Figure 2.3.1: Assessing intact egg with visible-infrared transmittance spectroscopy [3]

Since in this very special case, eggs are already broken, simplified and different methods can be considered. Initially, two ideas were researched by author of this thesis:

- Proximity sensor utilization
- Using laser / narrow light beam

Both methods would identify the intact yolk basing on its height and would not require much processing (an Arduino Uno or other ATMega based circuits might be sufficient). Nevertheless, both of them had to be rejected, since intact yolks presence in the tested batch is not sufficient for batch accepting – absence of egg yolk parts is also required (see subchapter 2.3).

Finally, using images of the batch obtained from camera and processing was considered. Such camera would be mounted just after the cracking module of RZ-1 machine and make decision whether or not the product batch looks like the one that is expected. This approach is very similar to the currently used one - the expert looking at the egg and making decision - except for the fact, that it would be fully automatised. Due to the potentially low price (camera and ARM computer price don't exceed 100$ in retail price) and no need for extensive modifications of RZ-1 structure, this method was chosen as most promising option for further research.

2.4 Detecting batch state with camera image

When it comes to assessment of camera image, 4 approaches are widely considered:

1. Very easy scanning with photo sensors or very low resolution cameras
2. Computer graphics preprocessing and segmentation accompanied with assessment of a parameter (number of pixels, number of feature points, percentage of one region that fills another bigger region and similar)
3. Machine learning approach
4. Neural networks which are used as a special case of either machine learning process or a part of it

First case was considered, since it could be implemented on Arduino Uno, other ATMega based platform or real-time systems working on PLC controllers such as LOGO! 7, which are often used in factories due to their reliability (apart from Stuxnet-class worms, that has been firstly

observed in 2010, no other security threads are believed to exist). Unfortunately, the problem is more complex than simply deciding whether the product is present or not; or whether the product is yellow or not, thus more advanced processing is required.

Second case will be expanded in the next two chapters and has been chosen as the method that will be implemented and tested in this thesis. Last two approaches will be tested if the 2nd approach proves to be ineffective.

It is worth mentioning, that for (3) two machine learning algorithms were tested:

1. Support Vector Machines - a statistical method that classifies lineary separable data to exactly two classes (i.e. proper and improper egg mass)
2. Haar Features Cascade - a method that applies series of so called weak classifiers (classifiers that are barely better than randomness) on the image to decide whether or not an object is present on it.

Unfortunately both approaches were rejected, since the amount of obtained testing data was insufficient to teach them and while the methods are effective in locating the particular object on image, they might not be the most suitable ones for comparing the amount of damaged yolk chaotically distributed on the egg batch.

(4) Case was considered and the author built a multilayered perceptron network working with MNIST database in order to better understand the method. Nevertheless, due to insufficient amount of sample data, neutal networks approach was abandoned.

3 Batch state detection system design

3.1 System structure

The system consists of (see figure 3.1.2):

- Camera module that captures images of cracked egg batches
- Processing device - an embedded computer (i.e. ARM based platform)
- Signal interface (i.e GPIO pins) on processing device

A camera will be placed on top of Separator module part (see figure 2.1.3), where cracked eggs are transported by sliding to the separating part.

The processing device uses the camera frames, to make a decision whether or nor the egg batch visible on the frame is acceptable, or represent a badly cracked egg. A signal will be send to element that will either remove egg batch from the processing plant or let is pass further (see figure 3.1.1). Development of this physical element is done by Ovo-tech company and is not analysed in the thesis. Ovo-tech declared, that having +5V signal for positive batch, and -5V for negative is enough for them to work on their part.

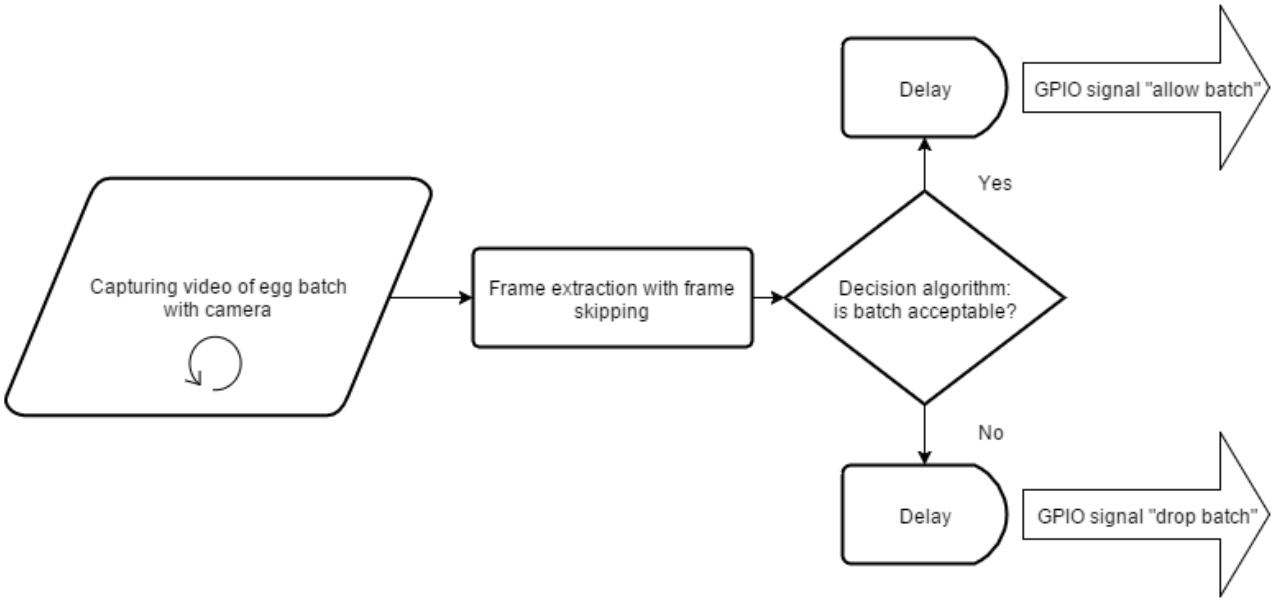

Figure 3.1.1: System workflow

Figure 3.1.2: Physical system structure - camera and processing device - Raspberry Pi 2 (unmounted from RZ-1 machine)

3.2 Hardware and operating system choice

Among over 20 available ARM systems, presented below were most interesting as possible to use in the system:

- Raspbian - a standard, default Raspberry Pi system in "Jessie" distribution.
- Minibian - a Raspbian image that is resource optimised. Its current Raspian "Jessie" based version boots in 14secs, use only 29MB RAM and takes 451MB (compared to 1,2GB in original version). GUI, application clients, office suite and other non-necessary packages have been removed. The image is targeted for embedded or server applications (NAS, Web server, electronic applications).
- Arch Linux ARM
- Gentoo
- SliTaz

- Windows 10 IoT Core - embedded solution developed by Microsoft. Guarantees extensive technical support.

Largest reliability and performance benefits are expected from using realtime-capable systems. Following solutions are available or developed for Raspberry Pi:

- RISC OS
- Xenomai
- PREEMPT_RT kernel patch and High Resolution Timers[24]
- ChibiOS/RT[25]
- RODOS - an Open Source kernel project developed by the German Aerospace Center and Prof. Montenegro's University[26]

Nevertheless advanced image processing, GPU access, utilisation of existing graphics libraries might be not available or require amount of effort disproportional to those benefits. Also, external real-time clock (such as AD9850 Pulse generator) would be required.

The decision was made, to use Raspbian as the operating system, because of its documentation availability and good community support.

Using alternative system operating on Raspberry Pi 2 platform might improve image processing performance. Therefore, more frames per second could be analized, more filters can be applied and higher resolution pictures could be used. Porting to one of the systems mentioned above will be considered in the future, during the process of designing mass-produced system version.

Following platforms were considered, to be used as processing device and host chosen operating system:

- Raspberry Pi 2 - featuring 900 MHz quad-core ARM Cortex-A7 and 1GB RAM, Broadcom VideoCore IV GPU, 4 USB ports, an ethernet port, 40 GPIO ports and camera module socket, its the most available of embedded pc ARM platforms.
- Arduino Uno R3 accompanied with DS1307 Real Time Clock - considered, but for now the image processing is to extensive in terms of image processing. Having extremely cheap clones (4-20$ on retail market) that features often stronger ATMega units is an advantage although.
- Intel Galileo Gen 2 - based on Intel Quark 32-bit SoC X1000 processor, provides pins-compatible with shields designed for the Arduino Uno R3, what makes communication with other Ovo-tech machine modules easier and allows use of the mentioned above RTC.
- PandaBoard - the OMAP4430 features 1 GHz dual-core ARM Cortex-A9 MPCore CPU (a big improvement compared to Cortex-A7 CPU in Raspberry Pi 2), 304 MHz GPU, and 1 GiB DDR2 SDRAM. What is interesting, it provides a real-time clock, what makes it a candidate for real-time system (see section 8.1).
- BeagleBoard xM - based on dual, 1,5GHz clocked ARM Cortex-A15 + dual ARM M4, 532 MHz GPu, often used with RISC-OS, having cheaper clones with more RAM.It is considered a stronger Raspberry Pi alternative.
- Nvidia Jetson TK1 - build on quad ARM Cortex-A15, NVIDIA Kepler 192-cores CUDA GPU, 2 GiB RAM, optimised for OpenGL and OpenCV, equiped with SATA connector. TK1 is the strongest available unit, what makes it also the most expensive one.

Raspberry Pi 2 was chosen, due to similar reasons that Raspbian was picked. Similarly to Raspian, Raspberry Pi 2 may be replaced with different platform in later design phases, when a prototype is operational and tested.

3.3 Detection algorithm

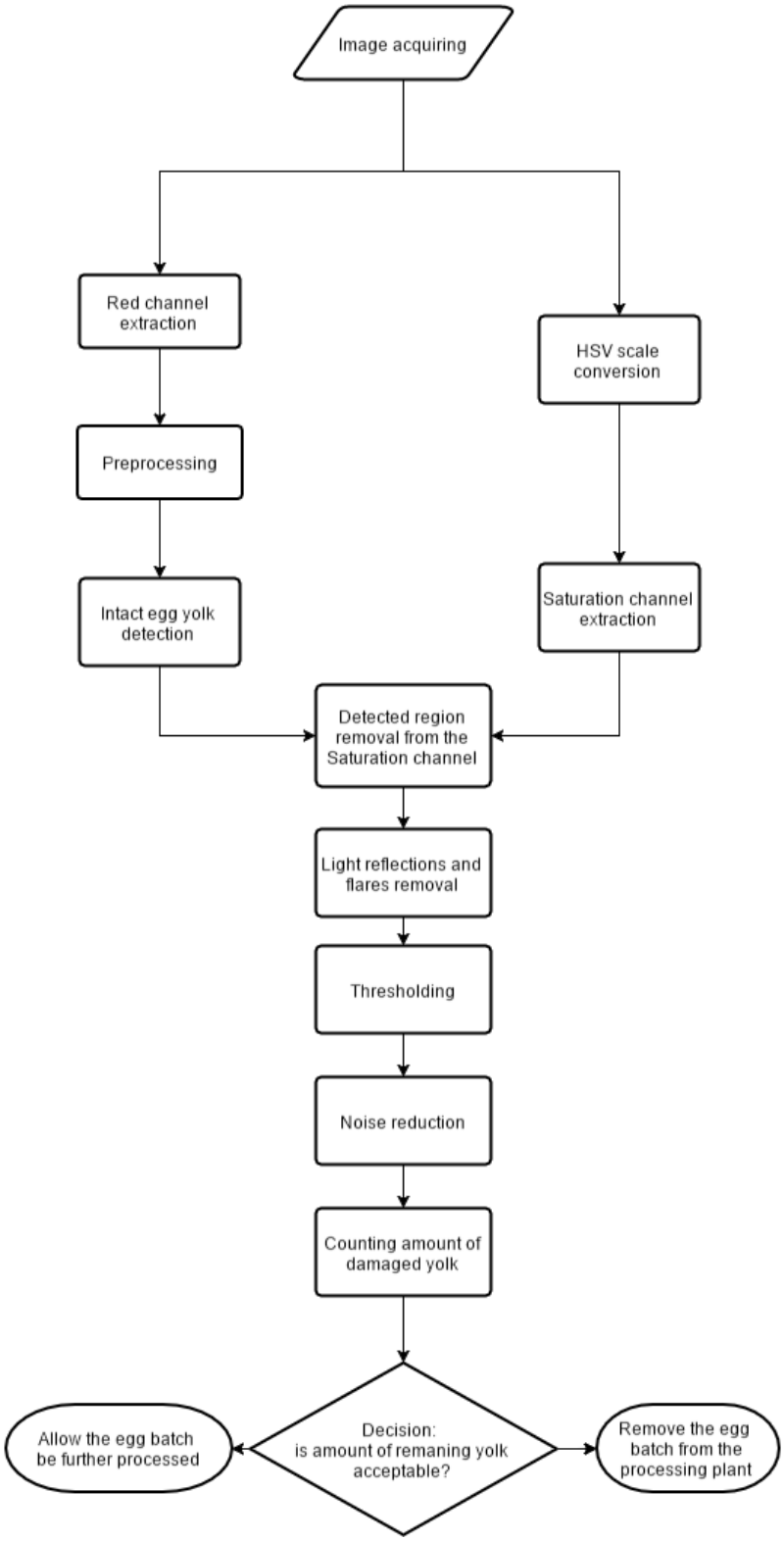

Figure 3.3.1: Flowchart showing the way that detection algorithm operates

Detection algorithm operates on frames captured with the camera module (see figure 3.3.1). Two copies of a frame are processed. The first copy is a subject of following transformations:

1. Single channel is extracted from the image. Its state is passed further as image, along all the steps.
2. The channel is a subject to preprocessing, that eliminates the unimportant data and reduces the data dimensionality. Amount of noise is reduced.
3. The image is segmentated, and the shape recognition methods are used to detect existence, position and size of the egg yolk.
4. Independently, other copy of the image is converted to different color scale.
5. Only one channel is extracted from the image
6. The information from first copy is used to remove intact egg region from second copy. The first copy is no longer needed and its resources are freed.

Obtained image (with intact eggs excluded) is processed in a following way, to asses the amount of remaining broken egg yolk:

1. An attempt is made to filter out the light reflections and flares from the image. Hopefully, during the further development, above artefacts can be removed in a physical way, by mounting and lighting the device in a very specific way. For now nevertheless, this step is necessary.
2. Image is segmentated with thresholding.
3. Noise (defined as small blobs) is deleted.
4. The obtained image is expected to contain only unwanted, damaged egg mass. Amount of pixels containing it is counted.
5. Decision is made basing on the obtained amount. It takes a form of signal sent to physical element that either removes the egg batch or let it through.

The best obtained combination of preprocessing, segmentation and grading parameters will be used in a final algorithm that will be implemented in Raspberry Pi 2 device with camera module, and than tested in the Rz-1 processing plant.

3.4 Software choice

Decision have been made, to utilise OpenCV computer vision library for batch state recognition. OpenCV is designed for computational efficiency, with strong focus on real-time applications. It has been written in optimised C and takes advantages of multicore processors. It provides simple-to-use interface that enables building sophisticated computer vision solutions. The library features more than 500 functions that process images, a general Machine Learning Library (including Multilayered Perceptron Networks). It is used since 1999 by major institutions such as Stanford University and is maintained and developed by specialists from companies such us IBM, Microsoft, Intel, Sony, Siemens, and Google.[8]

First recognition attempts were done by the author of thesis using Android 4.3 running on Samsung N7100 Galaxy Note II smartphone, utilising its build-in camera. The initial code was written with use of Java distribution of OpenCV library, as well as Android SDK. Sadly, the performance of developed pre-prototype was very poor and a single frame was processed in more than 3 seconds. Another version was built in a way that most resource consuming operations (such us Hough Circle Transform and all array operations) were rewritten in C language. For that purpose original C version of OpenCV was utilised, as well as NDK (Native Development Kit for Android).

Nevertheless, author of this thesis, decided that despite Android availability for embedded platforms, it is no a platform suitable for factory tasks, due to big overhead, long booting and variety of functions that will not be utilised.

Raspbian Linux "Jessie" distribution was picked as the adequate one to host recognition algorithm accompanied with Python 2.7 language. Python is well supported on Raspian, it produces small amount of code, and can be executed fast in terms of computational time if the specific C-written functions are called.[9] For author, a case occurred, that 39-lines code snippet that recognise nested regions of image, written in C took only 6 code lines after rewriting it in Python.

4 Egg yolk detection

4.1 General idea

This section describes idea introduced in 3.3 subchapter (Detection Algorithm), which is detecting the egg-yolk on the image depicting processed egg batch. Next, intact yolk image is removed from the image, so only remaining amount of damaged, fuzzy yolk can be measured (see. next chapter).

An egg yolk, when intact and fresh can be approximated with sphere which projects as circle-like shape on 2D image. Methods presented below enable to filter out the unnecessary noise, simplify the image and detect the yolk shape. While not all of them were used in the final detection algorithm, they were giving promising results on the early stage prototypes.

Parameters, which values have to be correctly chosen to effectively apply the used methods are indicated with **bold font** and summarised later on in Figure 6.4.1.

All the following methods treat images as 1, 8, or 24-bit encoded matrices. x and y denote modified pixel position (notion explained in next subsection).

4.2 Extracting color space channels

In digital graphics color information is usually stored as a value from RGB (or BGR) additive color space (see figure 4.2.1). It means, that three numbers represent amount of red, green, and blue subcolors that combined constitute a visible color. For the sake of Detection algorithm

considered in this thesis, it is assumed that every one of this numbers takes values from 0-255 interval and is encoded by 8bits, what sums to 24bit encoding for every RGB pixel.

Alternatively from RGB scale, simplified color space can be used: a grayscale image is made of pixels encoded by a single number that denotes the colors distance from being completely black; the higher is the number, the closer to being white it is. 8bit encryption has been chosen for grayscale images.

There is also an option to process black-and-white image with 1bit encoding - 0 picked for black, 1 for white - what is denoted as binary color scale.

A channel in this context is a grayscale image that represents intensity of single subcolor in RGB space. For example, a red channel is an image, in which 255 constitutes red pixel, and 0 means that pixel has no red subcolor.

It is suspected by the author that using a single channel instead of commonly used RGB image may is more efficient and some unnecessary information in dropped. Particularity, an egg yolk that is visually yellow or red my be favoured in red or green channel.

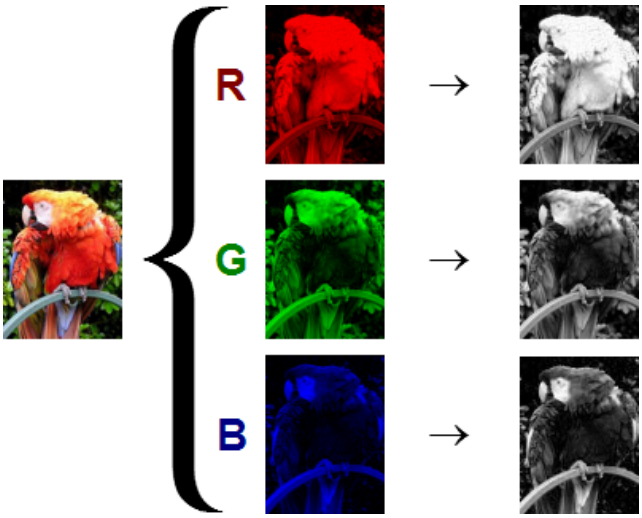

Figure 4.2.1: Three channels of RGB image. Src: www.commons.wikimedia.org

4.3 HSV scale conversion

Extending assumption from previous subchapter led author to believe that using a color space different than RGB might be more effective in recognition process. It might occur, that the problem of assigning color values associated with egg yolk is either linearly separable or close to being such, and thus extracting only those pixels that constitute egg yolk might be possible.

HSV cylindrical color space is known as being exceptionally suitable as a preprocessing method applied prior to image segmentation. [6] In HSV, pixel color is encoded as Hue (angle of a point on a cylinder), Saturation (a radius of a point) and Value (height of the point).

Hue is defined as "the degree to which a stimulus can be described as similar to or different from stimuli that are described as red, green, blue, and yellow"[7]. Saturation stands for intensity of a particular hue. Value is also defined as darkness of the color and is often treated as reverse grayscale channel (see figure 4.3.1).

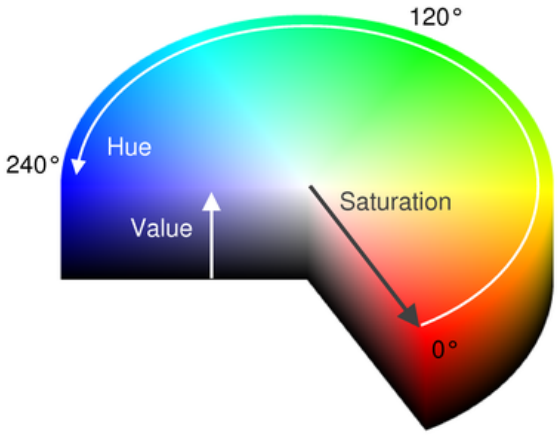

Figure 4.3.1: HSV Cylinder. Src: www.doc.qt.io

4.4 Circle Hough Transform (CHT)

Circle Hough Transform is an algorithm used for retrieving 3 parameters defining a circle-like shape on images[10] : - (xc, yc) - position of shapes center - r - its radius

To understand the way CHT works, simplified case is considered: The image is 1-bit encoded; the radius r and position (xp,yp) of a point lying on the searched circle are known. Potential circles space is created by treating points on a circle of radius r centered in (xp,yp) as centers for potential circles (see figure 4.4.1). For every potential circle, number of intersecting points is counted. The potential circle with largest number of intersections is chosen as the best fitting one.

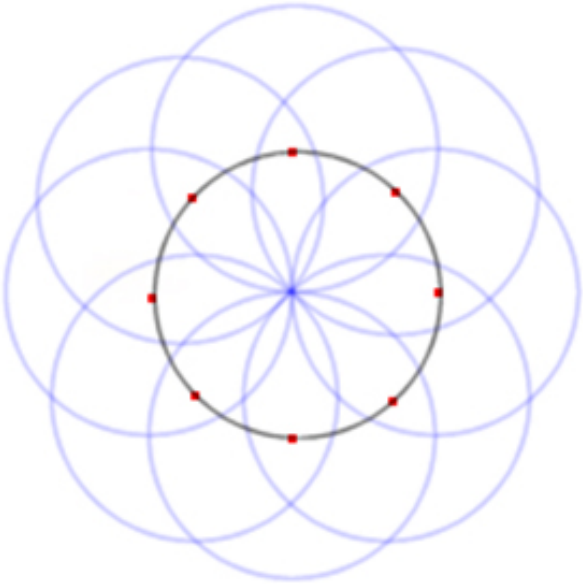

Figure 4.4.1: Potential circles space[11]

In the full cases, such procedure is repeated for every image point, and the intersections numbers are stored in accumulation matrix (see figure 4.4.2).[12] To reduce computation time and false detection rate, **minimum distance** between image centers is given as a parameter. Threshold value **param2** is defined as minimum number of intersections to treat circle candidate as detected circle.

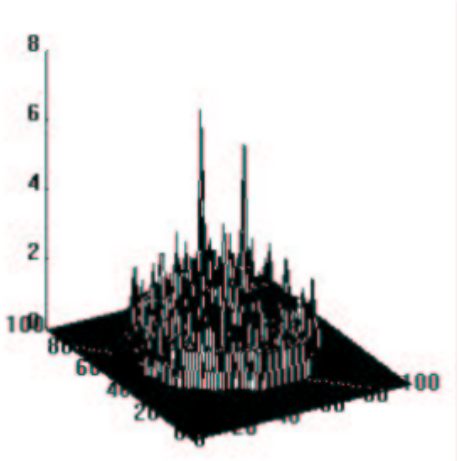

Figure 4.4.2: Accumulation matrix with two good circle candidates[12]

If r is unknown, min-radius and max-radius parameters are defined, and the algorithm repeats the procedure for all possible integer r in range of above parameters. OpenCV utilizes Canny edge detector for finding the potential intersection points, thus operates not only on 1-bit images, but also in 8-bit grayscale and 24-bit RGB images and provides smaller number of false positives.[13] **param1** is introduced as higher threshold for Canny edge detector, while the lower threshold is set to 0.5 * param-1[14] .

4.5 Erosion and Dilatation

Erosion and dilatation are two morphological filters that enable to filter out unnecessary noise from the image. Combined, they leave the core information (biggest blobs) intact or amplified, while small elements are deleted. It is easily applied on binary images:
A ROI (region of interest, a rectangular part of image) moves through image pixel by pixel and is compared to previously defined kernel (binary matrix, usually symmetric with respect to both diagonals - see figure 4.5.1). **Kernel size** is a parameter that defines how strongly filter alters an image.

```
0 0 0 1 0 0 0
0 0 1 1 1 0 0
0 1 1 1 1 1 0
1 1 1 1 1 1 1
0 1 1 1 1 1 0
0 0 1 1 1 0 0
0 0 0 1 0 0 0
```

Figure 4.5.1: A diamond-shaped kernel[15]

The consecutive pixels of on ROI that matches kernel elements with value 0 remain intact. Pixels of ROI that matches kernel 1-valued elements changes to: 1-ns if at least one of them is 1 for Dilatation 0-s if at least one of them is 0 for Erosion.

Erosion gives the effect of ‘shrinking’ the objects (and if they are small enough, they are deleted). Dilatation makes the objects bigger (see figure 4.5.2).

Figure 4.5.2: Original image, eroded image, dilated image.[16]

When these methods are used multiple times, and convoluted, they filter out the noise. Using erosion and dilatation with the same kernel one after another is called opening. Perimeter Determination and Skeletonization are widely used contour detection functions which utilize erosion and dilatation[16].

4.6 Gaussian blur

Gaussian blur is obtained by convoluting pixel with gauss function values. It reduces noise, detail and filters out the high frequencies what makes it a low pass filter.[17] Gaussian blur drastically reduces number of false-positive Hough circles detections. It nevertheless should be coupled with sharpening to reduce false-negative cases.[18]

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Figure 4.6.1: Gauss function in two dimension[19]

Kernel size is given as a parameter to a function that precomputes kernel, that is later convoluted with ROIs on the image.

The applied convolution is defined as piecewise multiplying two matrices and than summing the obtained values:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = (1*a)+(2*b)+(3*c)+(4*d)+(5*e)+(6*f)+(7*g)+(8*h)+(9*i).$$

Figure 4.6.2: Convolution of 3x3 matrices[20]

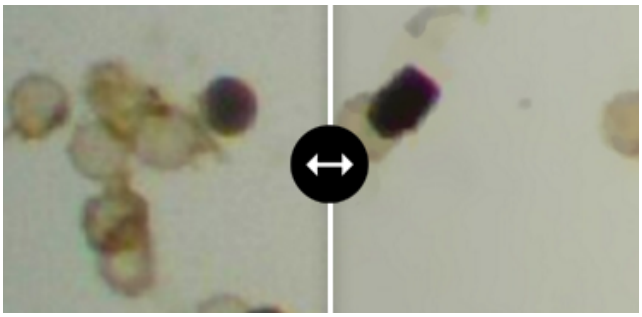

Figure 4.6.3: Screenshot of a tool that applies gaussian filtering on image[18]

4.7 Methods execution

The methods described above are used one after another as subsequent OpenCV library function calls in Python language:

```
def detect_yolks(frame):
    b,g,r = cv2.split(frame)
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    h,s,v = cv2.split(hsv)
    g = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # ...
    # 'c' variable - channel is chosen among the above
    c = cv2.GaussianBlur(c, (kernel,kernel), 0)
    c = cv2.threshold(c,threshold,255,thresholding_method)
    circles = cv2.HoughCircles(c,cv2.HOUGH_GRADIENT,1,dist,param1,
                               param2,minRadius,maxRadius)
    return circles
```

Figure 4.7.1: Python code snipped with OpenCV function calls

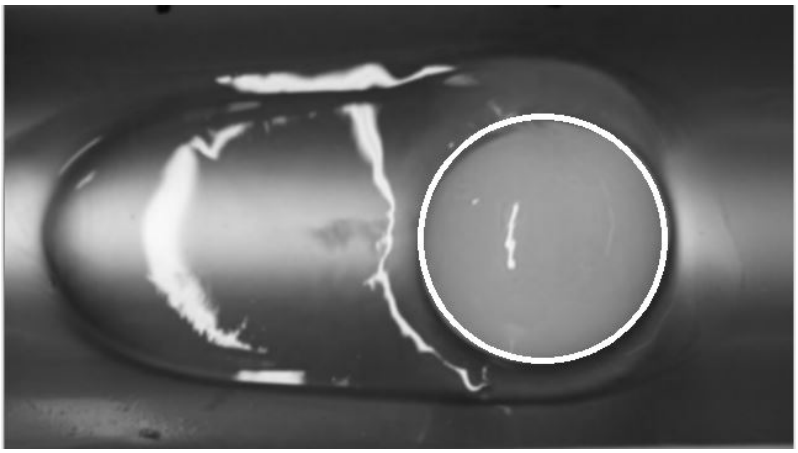

Figure 4.7.2: Properly detected intact egg yolk[18]

5 Contaminated product assessment

5.1 Subtracting yolk image

Subtracting previously detected images of intact egg yolks from the processed image outputs an image that holds only unwanted egg yolk, egg white and its surrounding. Therefore, the problem would be reduced to simple assessing the amount of remaining egg yolk and comparing it to a threshold value for acceptable bad yolk amount.

The initial implementation for this methods was supposed to be obtained by filling the image with empty circles detected in the previous chapter. Unfortunately, it has been experimentally tested, that egg yolk is reflecting some yellow light on it's surroundings and it is very hard to filter them out, thus a bigger area is subtracted. Instead of circles, empty rectangles are subtracted from the analysed image. The rectangles fill the whole image height (since eggs do not slide in parallel due to transportation slope design), and their width is equal to detected yolks diameter summed with an **offset** that covers reflecting light. The rectangles centers are equal to yolk centers.

5.2 Thresholding

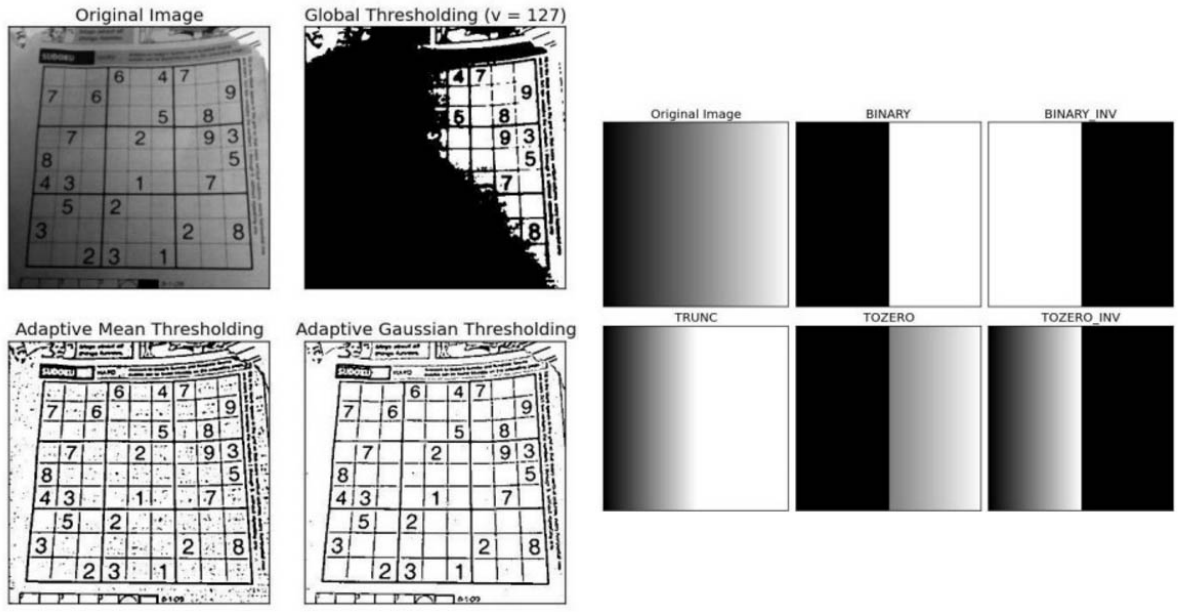

Figure 5.2.1: Different Thresholding modes generated with Metaplotlib library

Thresholding is a simple case of image segmentation (see figure 5.2.5). It can be used to convert one channel (in most cases grayscale) image to binary image.

The basic binary thresholding mode sets every pixel of destination matrix to:

- 0 if the source matrix value is smaller than **detection threshold** (integer parameter)
- 1 if the source matrix value is bigger than threshold.

The truncate version sets every pixel of destination matrix to

- threshold if the source matrix value is smaller than threshold, or
- leaves it unaltered otherwise.

Various modes and modifications are used depending on coding schema and colors distribution.

Finding an effective threshold value finding is a time consuming process and it often gives satisfactory results only for small set of images. A method called Otsu Binarization is often used to address this problem. Otsu Binarisation implements algorithm that automatically selects threshold value for each and every processed image.

Moreover using one threshold value for whole image may not work properly if the lighting conditions differ in different image areas.[21] Adaptive Thresholding utilize different threshold values for different ROIs. Two modes are often used: Adaptive Mean and Adaptive Gaussian. First compute threshold value as mean of neighbourhood values. Second uses weighted sum neighbourhood values.

On figures 5.2.2, 5.2.3, 5.2.4 author's attempts to find a thresholding method that distinguish egg yolk from its background are shown:

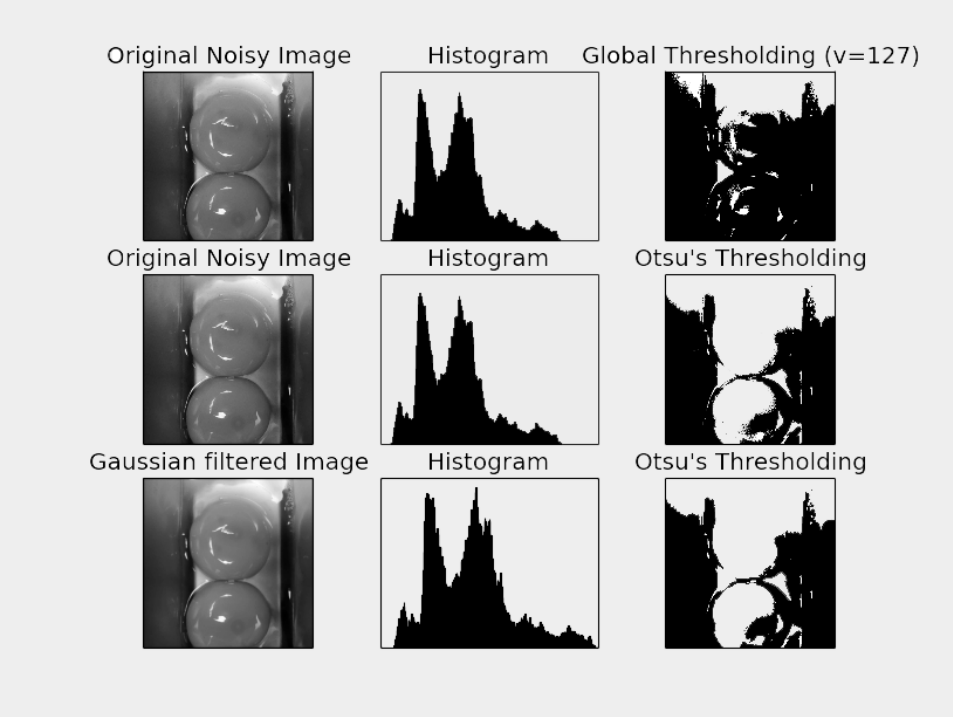

Figure 5.2.2: Different thresholding methods applied

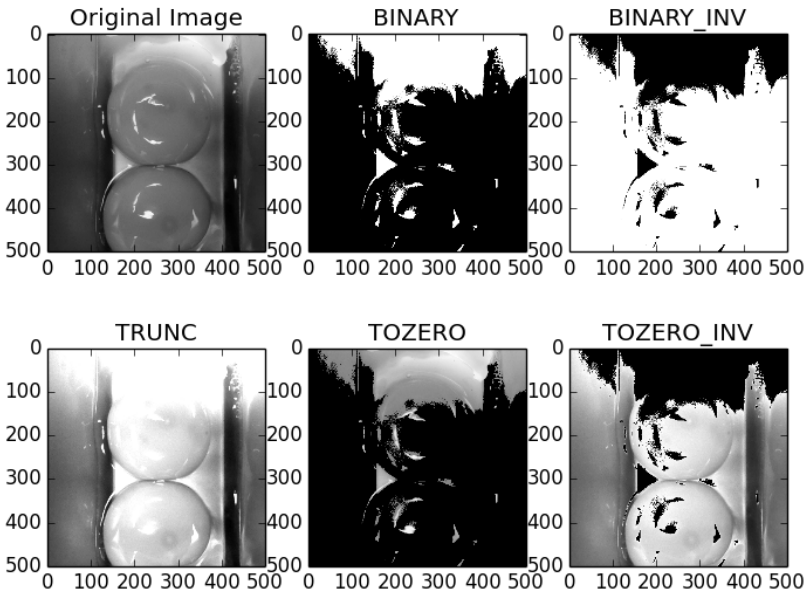

Figure 5.2.3: Inefficient thresholding of grayscale image

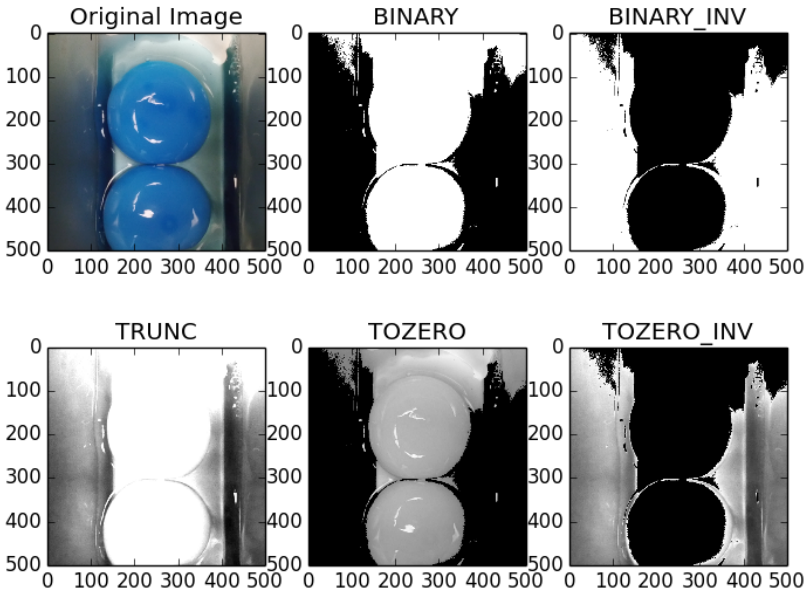

Figure 5.2.4: Much more effective thresholding of red channel image

For more general cases (i.e. multichannel images, assigning more colors) clustering method known as k-means is widely used.[22] K-means clustering was used as a supportive method for narrowing the effective threshold values.

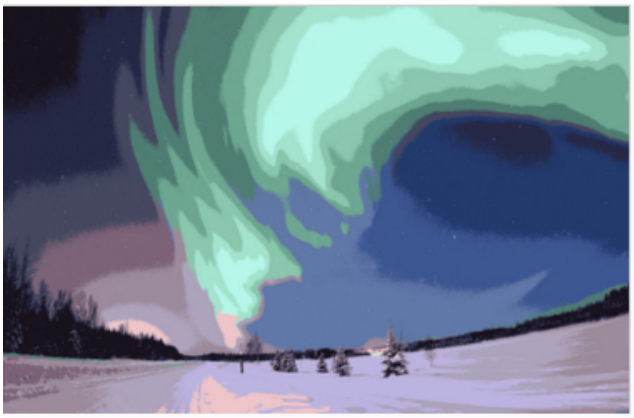

Figure 5.2.5: K-means clustering method for k = 16[23]

5.3 Methods execution

The methods described above are used one after another as subsequent OpenCV library function calls in Python language:

```
def grade_frame(frame, circles):
    b,g,r = cv2.split(frame)
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    h,s,v = cv2.split(hsv)
    g = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # ...
    # 'c' variable - channel is chosen among the above
    if isinstance(circles, np.ndarray):
        circles = np.uint16(np.around(circles))
        for i in circles[0,:]:
            cv2.rectangle(s_c,(i[0]-i[2]-offset,0),(i[0]+i[2]+offset,frame.height),0,-2)
    ellipse = cv2.MORPH_ELLIPSE,(kernel_size,kernel_size)]
    kernel = cv2.getStructuringElement(ellipse)
    c = cv2.morphologyEx(c, morphology_type, kernel)
    sum = np.sum(c)
    return 0 if sum > detction_threshold else 1
```

Figure 5.3.1: Python code snipped containing OpenCV function calls

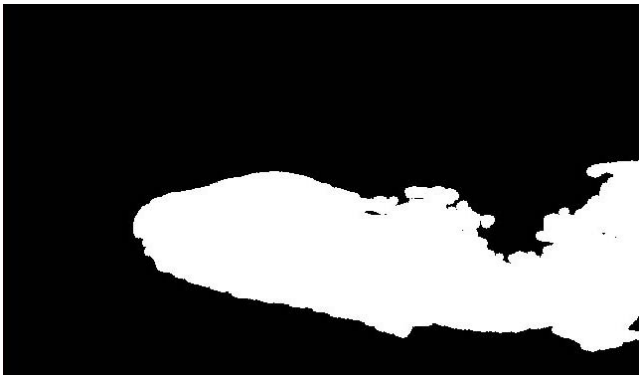

Figure 5.3.2: Properly segmented damaged yolk mass

6 Choosing detection system parameters

6.1 Manual parameters optimisation

There exist a set of parameters that define the mode, that previously explained methods operate as well as their insensitivity. Finding effective values of this parameters is a hard task, since for example CHT method was not able to found a single egg yolk with default numbers.

Few similar scripts were developed, that displayed over 500 sample pictures pictures (provided by Ovo-tech) for 0.3 second each in a loop. Various trackbars was displayed above the frames, with every trackbar corresponding to one parameter that is supposed to be chosen. Beside, the already transformed frames after transformations were shown. The expert was reviewing the output while changing the parameter values to obtain an image that segmentates the egg yolk properly.

Figure 6.1.1: Obtaining a set of sample pictures. Another set was obtained using Raspberry Pi camera.

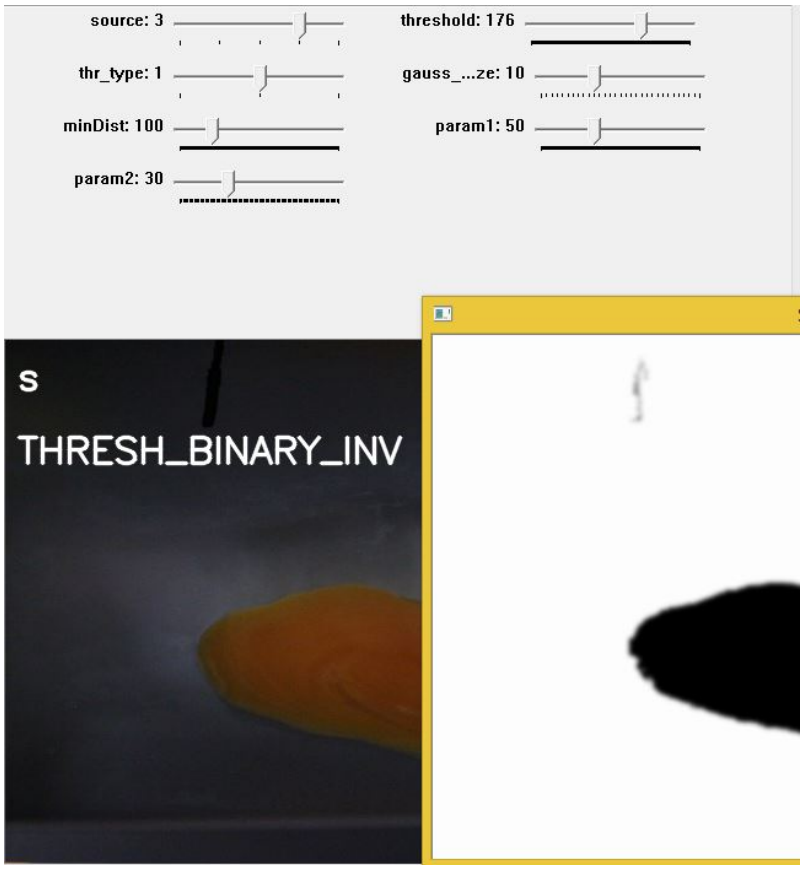

Figure 6.1.2: One of the scripts used for manual parameter interval finding

Creating GUI of parameter tuning script:

```
cv2.createTrackbar('source', 'Egg', 3, 4, nothing)
cv2.createTrackbar('threshold', 'Egg', 176, 255, nothing)
cv2.createTrackbar('thr_type', 'Egg', 1, 2, nothing)
cv2.createTrackbar('gauss_kernel_size', 'Egg', 10, 30, nothing)
cv2.createTrackbar('minDist', 'Egg', 100, 500, nothing)
cv2.createTrackbar('param1', 'Egg', 50, 150, nothing)
cv2.createTrackbar('param2', 'Egg', 30, 100, nothing)
# ...
param2 = cv2.getTrackbarPos('param2', 'Egg')
# ...
```

Unfortunately, while some parameters sets worked fine for some image sets, they failed with another ones. Nevertheless, the interesting intervals were obtained for each parameter. These intervals guaranteed that at least some of the graded images are segmentated in expected way. This method resulted in excluding most parameters values and vastly reducing the optimising problem size.

6.2 Automatic parameters optimisation

A decision was made, to automatise process of assessing the segmentation and detection efficiency to get to effective set of parameters. This process was done twice:

Firstly, the egg yolk detection process was optimised.
Secondly, the contaminated product detection utilising the previously optimised egg yolk detector was perfected.

To accomplish first objective, available images were divided into 3 subsets for first process:

- 0 intact yolks visible in a frame
- 1 egg present
- 2 eggs present

For the whole set, the recognition error was defined as sum of missdetections for every subset. A similar procedure was conducted for second objective: the same images were divided into following subsets:

- Contaminated batches
- Batches with either only intact eggs or no egg mass at all

Error was defined as false negatives (good batch recognised as bad batch) number summed with false positives number multiplied by 3. Multiplication by '3' weight was conducted on false positives, since keeping the product clean is more important for the client, that utilising all the egg mass.

A script was developed, that automatically loaded the images and graded them for different parameter values. This process was very expensive in terms of processing time, and the parameters range and step was chosen to fill a night (8h) of i7-3520M 8GB RAM computer work. It was computed in a following way: elapsed time of one full imageset testing multiplied by number of possible parameter sets. The process was repeated by the author multiple times with different parameter intervals discovered in a process descried in previous subsection. Various effort were made by the author to optimise imageset grading process, so bigger variety of parameters can be checked. Initially, files were open prior to every image evaluation, later on all the images were loaded once, and every processing (channel splinting, colorscale conversion) was conducted before execution of the code nested in multiple loops.

Multiple nested testing loops:

```
for param1 in xrange(10,80,5):
    for param2 in xrange(10,90,5):
        for kernel_size in xrange(1,30,2):
            for offset in (5,80,5):
                # ...
                grade_frame(detect_yolks(frame))
```

6.3 Overfitting

Overfitting is an unwanted effect that appears often in statistics and supervised machine learning. Overfitting means, that parameters of some model are fitting the specific modelled data case, rather than the general model.

In this case it means, that the parameters of image processing that were found, maximise the recognition efficiency for the data provided by Ovo-tech, while a new data (with different kind of eggs, different kind of noise or with images made in different conditions) can be classified in improper way.

Overfitting occurs mostly if:

- A set of sample data is too small
- A set of sample data is not diverse enough
- No regularisation or cross-validation methods are used

To reduce this effect, author used different dataset for getting the transform parameters, and different set (excluded 10% of data from every sample subset) for assessing the recognition quality.

During prototype in factory testing and further development this issue will be addressed in extended way.

6.4 Results

The following parameter values were obtained in a process of automatic parameters optimisation executed for on 195 sample photos that depict properly cracked egg and 388 depicting bad batches. The recognition accuracy maximised during optimisation process was 91,25%, and the recognition accuracy on the testing set of 57 images was 84,2

Parameter name	Domain of interest	Optimal value (yolk detection)	Optimal value (Contaminated product)
source	{bgr, r, g, b, grayscale, hsv, h, s, v)	r - red channel	s - saturation channel
param1 (HCT)	integers from [10:70]	60	N/A
param2 (HCT)	integers from [20:70]	25	N/A
minimum distance (HCT)	integers [100:500]	150	N/A
kernel size (erosion and dilatation)	odd integers from [3:35]	N/A	9
kernel size (gaussian blur)	odd integers from [3:31]	5	N/A
offset (deleting the yolk)	integers [0:50]	N/A	35
threshold type	{none ,binary, truncating, tozero}	none	binary
threshold adaptive method	{global, gaussian, mean, otsu}	N/A	global
threshold value	integers from {30-180}	N/A	170
detection threshold	integers from [900:30000]	N\A	2100

Figure 6.4.1: Parameter values obtained in automatic parameters optimisation

7 System implementation

7.1 Hardware and OS issues

Initially, hardware platform was connected to LG 27MP35 display through HDMI, iBox keyboard and mouse via USB. MicroSD SanDisk 8GB card (with EMTEC MicroSD Adapter)was used as a hard drive for operating system. TP Link WN725N USB WiFi adapter was used.

Device was powered from laptop USB port via microUSB cord, that has to be shaped to fit into Raspberry Pi casing (see figure 7.1.1).

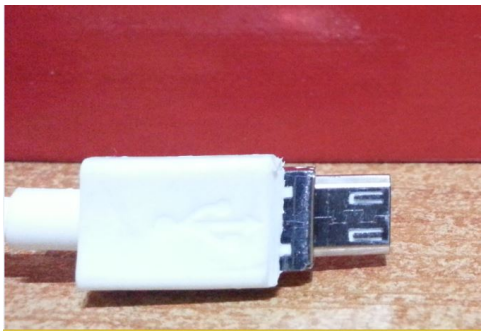

Figure 7.1.1: Cord shaped with a knife to fit the casing

Raspberry Pi Camera Rev 1.3 was picked to be used as a camera module (see figure 7.1.2).

- Fully Compatible with all models of Raspberry Pi
- Includes 150mm camera board ribbon cable
- 5MP Omnivision 5647 Camera Module
- Still Picture Resolution: 2592 x 1944
- Video: Supports 1080p @ 30fps, 720p @ 60fps and 640x480p 60/90 Recording
- 15-pin MIPI Camera Serial Interface - Plugs Directly into the Raspberry Pi Board
- Size: 20 x 25 x 9mm
- Weight 3g

Figure 7.1.2: Camera module features. Src: www.modmypi.com

To install Raspian system on the chosen platform, SD card had to be prepared. Diskpart windows tool has been used to delete all partitions and create a new one on SD card, since regular format was not enough to clear old Raspian installation - only 700 MB out of 8GB could be reclaimed that way. The card partition was formatted in fat32 file system. Later on, OS installer automatically partition it to both fat32 and ext4 filesystems partitions. Clean "Jessie" - "NOOBS" distribution in .zip archive was copied to drive.

Next, OS setup was conducted on device. Standard updates were conducted after WiFi network setup:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo rpi-update
```

Unfortunately, the screen was either blinking or had unreadable 320x240 resolution on 27MP35 display, and on tested L227WT display and EB-1761W projector displayed either 'out of range' error, or no image at all. Commenting the

```
hdmi_mode=16
hdmi_group=1
```

lines in boot/config.txt file and rebooting enabled devices to work properly in VGA mode.

Later on, the device was accessed via Putty SSH using either router with DHPC enabled or direct Ethernet to PC computer using static 169.254.1.1 ip reserved for direct devices connecting.

Xming software with lxsession was used for x11 forwarding - displaying the OS and GUIs on larger PC, so the egg flow could be viewed.

7.2 Python and OpenCV configuration

Downloading and building OpenCV with all its dependencies was along process (operations elapsing over 8h, not including the authors involvement). Following commands were executed:

```
$ sudo apt-get install build-essential cmake pkg-config
$ sudo apt-get install libjpeg8-dev libtiff4-dev libjasper-dev
   libpng12-dev
$ sudo apt-get install libgtk2.0-dev
$ sudo apt-get install libavcodec-dev libavformat-dev
   libswscale-dev libv4l-dev
$ sudo apt-get install libatlas-base-dev gfortran
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python get-pip.py
$ sudo pip install virtualenv virtualenvwrapper
$ sudo rm -rf ~/.cache/pip
```

Following lines were added to /.profile file:

```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
```

Than, following commands were executed:

```
$ source ~/.profile
$ mkvirtualenv cv
$ sudo apt-get install python2.7-dev
$ pip install numpy
$ wget -O opencv-2.4.10.zip http://sourceforge.net/projects/
   opencvlibrary/files/opencv-unix/2.4.10/opencv-2.4.10.zip/
   download
$ unzip opencv-2.4.10.zip
$ cd opencv-2.4.10
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/
   usr/local -D BUILD_NEW_PYTHON_SUPPORT=ON -D
   INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D
   BUILD_EXAMPLES=ON ..
```

```
$ make
$ sudo make install
$ sudo ldconfig
$ cd ~/.virtualenvs/cv/lib/python2.7/site-packages/
$ ln -s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so
$ ln -s /usr/local/lib/python2.7/site-packages/cv.py cv.py
$ workon cv
$ python
```

7.3 Code implementation

Below, the most crucial parts of Python code implementing the Detection algorithm which development was described in chapters 4, 5 and 6 is presented. The code is stored in detector.py file, and is automatically run when system starts. It has been achieved by appending a "sudo python detector.py" line at the end of /etc/rc.local file.

Firstly, used Python libraries are loaded. Picamera is used for communication with Raspberry Camera; numpy, itertools and cv2 are used for image matrices processing; os handles I/O operations, and time is used when performance is measured.

```
#library used for raspberry camera interface
from picamera import PiCamera
#used for storing and processing the camera frames
from picamera.array import PiRGBArray
#used for execution time measurements
import time
#opencv image processing library
import cv2
#library used for processing images as matrices
#works faster than Python lists
import numpy as np
#method used for array comprehension operations
from itertools import izip
#used for saving or loading images from storage
import os
```

Figure 7.3.1: Importing dependencies

The most important function, that recognise good and bad batch at obtained camera frame:

```
def process_frame(frame, thresh, param1, param2, min_radius, max_radius,
        min_distance, gauss_kernel_size, dilate_kernel_size, detection_threshold):
    #obtaining red channel - 3rd element of tuple returned by split() function:
    red = cv2.split(frame)[2]
    #converting BGR image to HSV scale and obtaining Saturation channel:
    saturation = cv2.split(cv2.cvtColor(frame, cv2.COLOR_BGR2HSV))[1]
    #thresholding with chosen method:
    saturation = cv2.threshold(saturation,thresh,255,cv2.THRESH_BINARY)[1]
    #gaussian blur for better yolk detection:
    red = cv2.GaussianBlur(red, (gauss_kernel_size,gauss_kernel_size), 0)
    #detecting yolks with Hough Circles Transform:
    circles = cv2.HoughCircles(red,cv2.HOUGH_GRADIENT,1,min_distance,param1,
                               param2,minRadius,maxRadius)
    #if circles are found, they are exlcuded from saturation channel:
    if isinstance(circles, np.ndarray):
        circles = np.uint16(np.around(circles))
        for i in circles[0,:]:
            #rectangle with offset excluded to delete yellow light reflected from egg yolk:
            cv2.rectangle(saturation,(i[0]-i[2]-offset,0),(i[0]+i[2]+offset,480),0,-2)
    #dilatation and erosion used as image opening to reduce noise:
    saturation = cv2.morphologyEx(saturation, cv2.MORPH_OPEN,
        cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(dilate_kernel_size,dilate_kernel_size)))
    #summing the remaining egg yolk:
    sum = int(np.sum(saturation)/255)
    #decision whether or not batch is acceptable:
    return 0 if sum > detection_threshold else 1
```

Figure 7.3.2: Recognition function

Below, the way that Raspberry Pi camera is handled is shown:

```
# initialising the camera:
camera = PiCamera()
camera.resolution = (640, 480)
# framerate (frameskipping) that works without delays:
camera.framerate = 10
rawCapture = PiRGBArray(camera, size=(640, 480))
#warmup time:
time.sleep(0.1)
#grabbing subsequent camera frames:
for image in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
    #image.array is a numpy array, the same type that opencv processes:
    frame = image.array
    process_frame(frame, ... ) #values of parameters from Figure 6.4.1 are input here
    #clear the image container:
    rawCapture.truncate(0)
```

Figure 7.3.3: Camera frames grabing

8 Conclusions and further research

Work on the solution developed in this thesis will be continued further on by the author and Ovo-tech company. The following alternative approaches and improvements in detection system will be taken into consideration:

8.1 One month prototype testing and environment of work tuning

The prototype will be attached to the RZ-1 machine operating at Ovo-tech headquarters for a month. During that time, led lights will indicate proper and improper egg batch. Recognition efficiency is now > 80%. Two cases may occur, when the device is operating in working environment:

1. Recognising efficiency will rise dramatically, since the camera position will be fixed and the background (separation module slope) will have uniform facture and color. The data provided by Ovo-tech for prototype construction covered various situations and backgrounds that will be invariant in factory application.
2. Recognising efficiency will drop, due to varying egg quality, color and shape (the overfitting problem).

Two things will be done to improve image material quality registered by camera:

- Polarizing filter will be attached to the camera. It should eliminate or drastically decrease amount of white light reflexes. Those reflexes that make yolk appear as non-yolk at segmentated image.
- Lighting will be uniform due to covering camera and a fragment of transportation slope in a box-like container with invariant, non-shadowing lights inside of it.

Author and Ovo-tech expect a large efficiency increase after those modifications.

8.2 Testing new approaches

Two approaches have came up after the prototype development and are to be implemented and tested:

- Circularity testing - height of the segmented yolk flow can be treated as a circle diameter. Than, area of obtained circle can be compared to area of observed egg yolk. If the difference is significant, it indicates that observed blob is not a whole egg. Using a derived method for positioning the center of such circle might work better than methods used up to now if a noncomplete part of egg is visible at the image.
- Watershed algorithm accompanied with extracting peaks of Euclidean Distance Transform is believed to work better than contour detection and thresholding methods with touching or overlapping objects. It might provide efficiency boost in case of RZ-1 processing plant operating in modes faster than precise mode.[27]

8.3 Obtaining more image data and parameters tuning

Larger dataset of more different egg types will enable to better tune the processing parameters. Since there are more than 10 parameters to adjust and the number of operations that are required rise exponentially with parameters number and is multiplied by sample amount, optimising such multivariate function is a hard task. Genetic algorithms are known to be a good metaheuristics for such tasks and should give better results in much less computation time that brute force parameter testing. Author developed a program that implements genetic algorithm in Java, that optimises 1,2 and 3-variate functions to understand and learn the genetic algorithm concepts. Further on, a ready, popular library will be used for this purpose: a DEAP genetic programming library for Python.

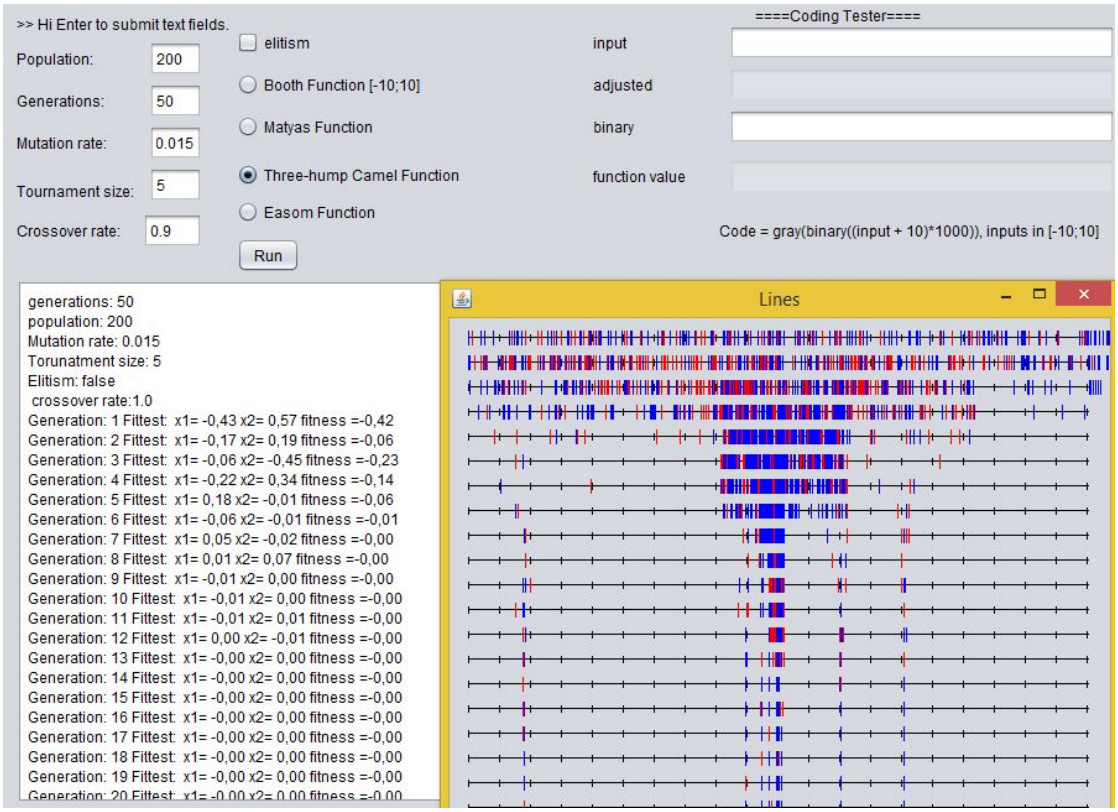

Figure 8.3.1: Screen of Java genetic function optimiser written by author of this thesis [15]

8.4 Server and logging

Gathering the data about egg quality, recognition performance and effectiveness is a possibility that has to be addressed. Raspberry Pi 2 platform is capable of working as a server and thus provide system manufacturer a remote access to the its operations.

Such process may generate additional benefit, which is deciding whether or not the processed eggs quality is good.

Ovo-tech reported cases of unreliable egg suppliers, that shipped old, misclassified eggs either purposefully or unwillingly to the bakery companies. Such situation would be detected by monitoring statistics of how big percentage of processed eggs are badly broken.

8.5 Conclusions

Following conclusions were obtained in the system designing, tuning and testing process:

1. Recognising egg-batch state using camera image is an adequate method of solving the problem of imperfect egg-white separation.
2. Simple image processing methods are cable of obtaining image recognition efficiency higher than 84% on this problem, even if lighting and camera placement conditions are unfavourable.
3. An operational prototype device was constructed and implemented. It will be tested and improved further in order to introduce it to mass production. Ovo-tech will be expanding its R&D section to obtain that goal.

Both the project goals has been achieved.

References

[1] Ghose T. *Could Science Hatch the Perfect Fake Egg?* LiveScience Web. 2013

[2] Berry D. *Egg product functional properties* American Egg Board Web. 2013

[3] Mehdizadeha S., Minaeib S., Hancockc N. H. ,Torshizid M. *Information Processing in Agriculture*, Volume 1, Issue 2 Print. 2014

[4] Deng X., Wang Q., Chen H., Xie H. *Eggshell crack detection using a wavelet-based support vector machine* Comput Electron Agric, Print. 2010

[5] Ketelaerea B.,Bamelisa F., Kempsa1 B., Decuyperea1 E., Baerdemaekera J. *Non-destructive measurements of the egg quality* World's Poultry Science Journal, Volume 60 / Issue 03, Cambridge University Press, Print. 2004

[6] Li N., Bu J., Chun C. *2002 International Conference on Image Processing*, Volume 2, The Institute of Electrical and Electronics Engineers, Print. 2002

[7] Fairchild M., *"Color Appearance Models: CIECAM02 and Beyond"*, Tutorial slides for IS&T/SID 12th Color Imaging Conference Web. 2004

[8] Bradski G. ,Kaehler A. *Learning OpenCV: Computer Vision with the OpenCV Library* O'Reilly Media Inc., Print 2009

[9] Gorelick M., Ozsvald I. *High Performance Python: Practical Performant Programming for Humans* O'Reilly Media Inc., Print 2014

[10] Opencv Dev Team *Hough Circle Transform* OpenCV 2.4.12.0 Documentation, Web. 2014

[11] Milbourne A. *Computers Counting Craters* Birkbeck College, Web. 2012

[12] Rhody, Harvey *Hough Circle Transform* Chester F. Carlson Center for Imaging Science Rochester Institute of Technology, Web. 2005

[13] Kapur S., Thakkar N. *Mastering OpenCV Android Application Programming* Pack Publishing, Print. 2015

[14] Opencv Dev Team *Feature Detection* OpenCV 2.4.12.0 Documentation, Web. 2014

[15] Opencv Dev Team *Morphology Fundamentals: Dilation and Erosion* MathWorks Documentation, The MathWorks, Inc. Web. 2014

[16] Opencv Dev Team *Eroding and Dilating* OpenCV 2.4.12.0 Documentation, Opencv Dev Team, Web. 2014

[17] Shapiro L. G., Stockman G. C *Computer Vision* Prentice Hall, Print. 2001

[18] Khvedchenya E. *How to detect circles in noisy images* Computer Vision Talks, Web. 2014

[19] Nixon M. S., Aguado A. S. *Feature Extraction and Image Processing* Academic Press, Print. 2008

[20] Lecarme O., Delvare K. *The Book of GIMP: A Complete Guide to Nearly Everything* No Starch Press, Print. 2013

[21] Opencv Dev Team *Image Thresholding.* OpenCV 2.4.12.0 Documentation, Web. 2014

[22] Barghout L., Sheynin J. *Real-world scene perception and perceptual organization: Lessons from Computer Vision.* Journal of Vision, Print. 2013

[23] Barghout L., Sheynin J. *Image segmentation* Wikipedia, Wikimedia Foundation Inc, Web. 2015

[24] Gupta V. *Patchwork Gaurantee spinlocks implicit barrier for PREEMPT_COUNT* The Linux Kernel Archives, Web. 2013

[25] Bate S. *ChibiOS/RT on the Raspberry Pi* ChibiOS EmbeddedWare Web. 2015

[26] Montenegro S., Dannemann F. *Real Time Kernel Design for Dependability* DASIA 2009 DAta Systems In Aerospace, Paper. 2009

[27] Rosebrock A. *Watershed OpenCV* PyImageSearch Course, Web. 2015

All the Web. sources have been checked for availability in November 2015. Web links are not included if articles are to be immediately found with search engines, which is consistent to MLN referencing convention. For modified web documents, date of last modification is shown.

If it has not been specified otherwise, figure images has been taken or prepared by the author of this thesis.