Determining the directionality of an object using computer vision

P6 Project
Group 620
Computer Technology & Electronic and IT
Aalborg University
25 May 2022

AALBORG UNIVERSITY
STUDENT REPORT

Sixth Semester Students, Computer technology & Electronic and IT
ComTek & EIT
Frederik Bajers Vej 7
9220 Aalborg

Title:

Determining the directionality of an object using computer vision

Project:

P6-project

Project Period:

February 2022 - May 2022

Project Group:

620

Participants:

Kazim Kerem Kocan
Khadar Guled
Laurits Winter

Group supervisor:

Thomas B. Moeslund

Co-supervisor:

Simon Buus Jensen

Abstract:

This report outlines the development of a system that is able to detect a cage, and determine the direction it is moving. The report covers topics such as package shipping, the procedure by which it is done, and the consequences when this fails. It also describes technologies pertaining to masking, morphology, histograms and edge detection.

The code implemented is split into the detection, and the identification of movement. During testing the code has been proven to be relatively consistent regarding detecting and determining , and well beyond expected levels of accuracy for a prototype, however not sufficient for a practical application. the shortcomings causing this lack of accuracy, as well as means of fixing them is outlined in the discussion. The project has succeeded in showing the feasibility of using vision technology to identify, and extract motion data from objects.

Page Count: 54
Appendix: 5
Finished 25th May 2022

Foreword

This report is written by the project group 620 during the sixth semester project at Aalborg University.

We thank Michael, Gert, Christian and Christian from Lyngsoe Systems for their consistent and helpful cooperation, as well as for providing the facilities necessary for the project.

We thank Thomas Moeslund our supervisor, and Simon Buus Jensen our co-supervisor, for their frequent and detailed feedback and intellectual sparring.

We praise Simon Buus Jensen for going above and beyond what is expected, and dedicating vast sums of time to our benefit.

Kazim Kerem Kocan

Khadar Guled

Laurits Winter

Acronyms

AHE Adaptive Histogram Equalization

CLAHE Contrast Limited Adaptive Histogram Equalization

EM Electromagnetic

FPR False Positive Rate

ISO International Organisation of Standardisation

KNN K-Nearest Neighbors

MOG Mixture-of-Gaussians

RFID Radio-frequency identification
ROI Region Of Interest

SE Structuring Element

TPR True Positive Rate

Contents

1 Introduction

The Danish post services handle in excess of 150 million packages a year. When distributing so many packages, some are bound to get lost in the process. Finding and re-sending these packages costs the post services an absorbent amount of money, and as the scale of the system slowly grows, so does this cost.

Currently, the post services are working with Lyngsoe Systems to keep down this cost, by preventing packages from getting lost in shipping. Their current approach is to attach an RFID (Radio-frequency identification) tag to the rolling cages in which the packages are transported in, and an RFID reader above the gates for distribution centers. If a cage travels through a port, these sensors will pick up its ID, and it will be logged.

This method poses a few major issues, namely a problem with radio noise, and with the inability to determine whether the cage is entering or leaving the building. The noise issue is founded on the fact that the RFID sensor has a long-range, and will pick up all nearby cages with a strong enough signal. It can therefore pick up cages that pass by the gate, without going through it, cages that are just parked in the vicinity, and cages that have not had their tags removed after being emptied. The RFID sensor is also blind to any cage which is not tagged. Lyngsoe Systems has attempted to address the issue of determining the directionality by looking at the phasing of the return signal from the RFID tag on the cages. While this does produce results, it is not very reliable. Based on the issues with the RFID sensor, as well as the desire for their product to have higher levels of reliability, and interest in vision technology, they have proposed a bachelor project to AAU, centered around the possibility of utilizing computer vision to supplement the RFID sensor in the identification of cages, and determining the directionality of the cage.

Based on the proposed problem, an initial problem statement is formed:

How can vision technology be used to detect an object and how can the directionality of the object be found?

Problem Analysis 2

2.1 Magnitude of the problem

Package shipping is becoming more and more a regular routine for people globally. 70% of the Danish citizens between the age of 16 to 89 have shopped on the Internet in the previous three months, a number increased from 53% in 2011 [Statistics Denmark, 2020]. This type of growth is seen worldwide with over two billion people having purchased goods or other services online - this meant that the worldwide e-retail sales surpassed 4.2 trillion USD [Statista, 2022]. This growth is also paralleled in the worldwide package shipping business with 131 billion packages being sent in 2020 alone as shown in figure 2.1. It is estimated that the volume of global package shipping will be doubled by 2026 [Statista, 2021].

Figure 2.1. Graph showcasing the global package shipping between 2013-2020 [Statista, 2021]

The Danish Civil Aviation and Railway Authority (DCARA) published a similar report stating that the total amount of packages sent to or inside Denmark was approximately 170 million packages in 2020 - meaning that on average around 465.000 packages are sent daily [DCARA, 2020]. This large volume of packages means that it is important to be able to manage the packages efficiently and reduce the margin of errors that can happen during the process of package delivery. Thereby, an understanding of how package delivery companies manage the packages and the process of package delivery is needed.

2.2 The process of package sorting

A sorting center is a facility where packages are sent to sort them with regard to the destination place. The incoming packages are then sorted based on the region of the receiver and the sorted packages are inserted into rollable cages. When sorted, the packages are then shipped off to the correct destination place through ports. Packages are usually distributed from the sorting center to distribution centers based in the region of the receiver, which then sorts the packages based on the towns and then shipped to the local post office, which then has the responsibility to deliver the packages to the receiver as shown in figure 2.2. This method helps break the large volume of packages into manageable numbers and is therefore used by postal services and logistic companies. Thereby, most postal services have few sorting centers and many distribution centers located around the country [US Global Mail, 2022]. As an example, the largest Danish postal service Post Danmark A/S has four sorting centers in total (Taulov, Brøndby, Kolding, and Aalborg) and over 100 distribution centers placed around the country [SCM, 2014].

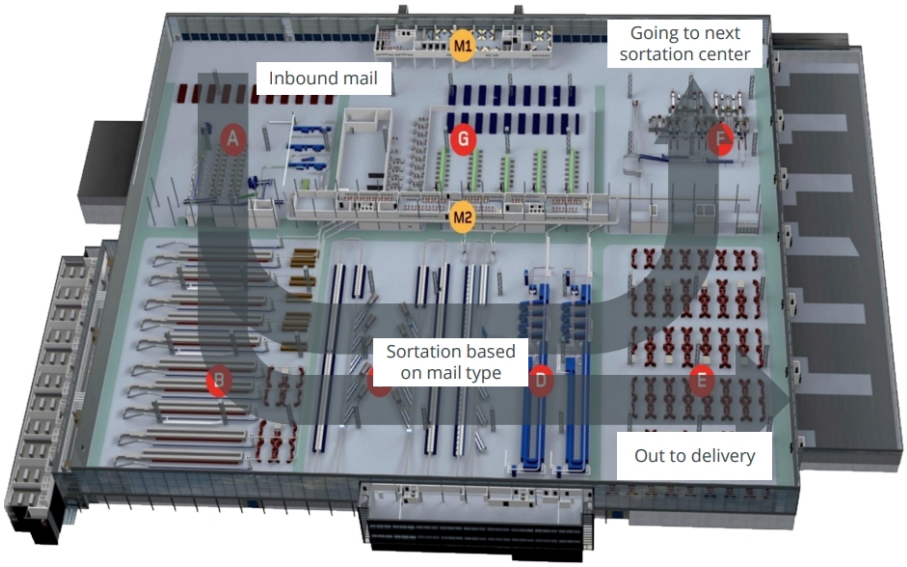


Figure 2.2. The flow of package through the sorting center

However, during the sorting process, packages might get incorrectly sorted and shipped into a different region or town than expected. In case of incorrectly sent packages, several administrative and operational processes are triggered. Firstly, the incorrectly sent packages need to be tracked, this is generally done as a communication between the sorting center and the local distribution center or post office, who have these packages. Then transport needs to be arranged such that the packages can be shipped back to the sorting center. When shipped back, the packages go through the sorting process once more, prior to shipping them to the correct destination place or distribution center.

The process is generally time-consuming, resulting in a delayed delivery time and calls from the consumer to the service center, which at worst can be reputation-damaging for the company. In Denmark in 2020, there was 11.810 complaints from customers to shipping companies due to either lost packages or unsatisfactory delayed delivery time, where approximately 54% of those complaints received compensation as a result

[DCARA, 2020]. Therefore, while this process is charged with expenses (i.e. arrangement of transport and sorting process), there might be other expenses in form of compensation, customer service etc. In their yearly report, Post Danmark A/S revealed that they spent 63 million DKK in compensation alone in 2020 [Post Danmark A/S, 2020]. However, the different reasons to the compensations are not given, and it should be assumed that reasons such as compensation due to damaged packages are included. Due to this, Lyngsoe Systems estimates that incorrectly sent packages cost the companies around 25 million DKK in Denmark, while in larger countries such as USA the estimated expenses are at 1 billion DKK. Thereby, due to the nature of the cost, there is a commercial interest in solving the problem.

2.3 Environment

As described in chapter 1 and 2.1, the desired product will be used in sorting centers. It is therefore important to account for key properties of the cage and diverse elements related to the environment in sorting centers and notice potential problems with regards to object detection.

2.3.1 Cage

As described in section 2.1, packages with a common destination place are inserted into the same rollable cage. Regarding color, there exist no standard regarding the coloring of the cage. One notable trait, however, is that cages generally have the color of steel/aluminum which is naturally close to silver/gray as shown in figure 2.3, while also having the possibility of discoloration either from use or misuse.

Figure 2.3. A typical cage used to distribute packages

A potential problem from a perspective of view of object detection is the glossiness or reflections of the cage. Especially since sorting centers use artificial lighting as described in 2.5, the cage might not have a homogeneous color. This can make it more difficult for a object detection system to detect the cage.

2.4 Basics of cameras

Based on the initial problem statement in chapter 1 and the potential consequences the problem can have as described in section 2.1, one of the key elements is the acquisition of images. Thereby, it is necessary to account for how a camera can capture light and convert it into an image.

2.4.1 Image acquisition

Based on section A.1, the basics of light and the range, human eyes can see, are described. This leads to the following question: How does one capture light to form an image? This can be answered by setting up an example as shown in figure 2.4.

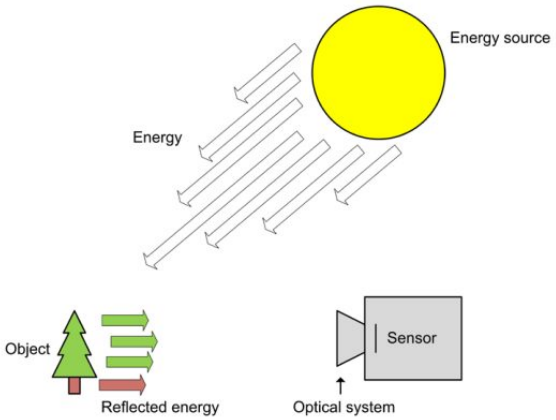


Figure 2.4. Overview of the general process of image acquisition [Moeslund, 2012, Chapter 2.1]

Firstly, for any optical system such as human eyes or camera lenses, to be able to pick up an image, the desired object in focus needs to be illuminated. The light comes from energy sources, who emits energy in form of light. The energy source can either be natural such as the sun, artificial such as a lamp or a combination of both. In the example shown in figure 2.4, it is assumed that the only energy source is the sun, which emits the light. When the light hits the object, it gets reflected in different directions. Some of the light will then hit the camera lens, which then hits the image sensor, where the image is formed.

The image sensor is generally used for digital cameras to detect the reflected light. Image sensors contains an array of cells commonly known as pixels. Each pixel detects and measures the amount of received light, which then is converted to voltage and by using an analog-to-digital converter, it is finally converted to a digital number. An important property of an image sensor is the size. A bigger sensor size will mean that it is possible to get a larger field of view and thereby more of the surroundings will be captured using the same camera lens compared to a small sensor size. The bigger sensor size also means that the pixel size will get larger, meaning that there will be less noise in the image and a better light sensitivity. However, the disadvantage in having a larger pixel sizes is a loss of details in the image.

Focus

It is a key parameter that the camera is able to capture the desirable object without focusing on unnecessary details in the background. This would avoid a larger processing time and is highly desirable. While working with cameras set in a fixed position, it is important to set up the camera lens using the distance from the object to the lens.

Field of View

Another key parameter for capturing a desirable image is the field of view, which is the observable area of the camera. The field of view depends on the placement of the camera together with camera settings such as focal length and size of image sensor. The desire of having a bigger or smaller field of view depends on the specific task: If it is desired to track an object where the route is known, it would be desirable to have a relatively small field of view since it narrows the image to the specific area where the object is. A higher accuracy might thereby be recalled compared to a bigger field of view.

2.5 Background Noise

In Danish sorting centers, the light condition must comply with the Danish standard DS/EN 12464-1 (Light and lighting - Lighting of workplaces - Indoor work places). The lighting conditions in a sorting center must be at least 300 lux. Lighting systems are also required to be designed such that the lighting conditions remain constant [Danish Standards, 2021]. Generally, in sorting centers there is no windows meaning that there exist no additional natural light except when the ports (described in section 2.1) opens. However, the ports open only if there is a truck ready to be loaded. Here, the natural light is at a minimum, since the trailer of the truck blocks any light from coming through. Therefore, in overall, the lighting condition in a sorting center can be described to be constant artificial lighting with no natural lighting.

In terms of object detection, the primary goal is to detect the cages which are loaded into given lorries. However, the environment contains variations of objects that will occasionally enter the frame such as forklifts, truck lifts, pallet trucks, individual packages and personnel generally in form of workers. These objects are **excess objects** and should be ignored by the object detection system. Since the workers interfere with the cage, one of the key points is for the object detection system to detect and differ the cage and the worker.

Finally, the data from camera could become a source for noise. This can happen due to the format of the video/image file and thereby the file compression as described in Appendix A.2. Since the object detection system is desired to work in close collaboration with the RFID system as described in chapter 1, the data from the camera should be streamed - in that case a loss of data could be expected.

2.6 Image Processing

Based on section 2.4 and 2.3, some background knowledge of cameras and potential problems in the environment regarding sorting centers have been described. Especially

based on section 2.4, it is important to get a basic understanding of how images can be processed and manipulated with. In this section, the basics of images and image manipulation are described.

2.6.1 Data interpretation and manipulation

Digital images are represented as two-dimensional arrays. An image is in these cases seen as a discrete function $f(x, y)$. A specific cell or dot in the image is called a pixel. Pixels present the values of the interpreted intensity from cells of the image sensor which is further explained in section A.2. Regarding the amount of bits for a single pixel, it is variable depending on the image type and color. Generally, a pixel contains 8 bits (equal to 1 byte) if it is in gray-scale (i.e. black-and-white). This means that there is $2^8 = 256$ different values. For a gray-scale image, the pixels are represented as a gray-scale value ranging from the interval from black (0) to white (255). In colored images, the pixels have three values representing the three primary colors, red, green and blue, hence the name RGB. The three values each contains generally 8 bits similar to the gray-scale image [Moeslund, 2012].

When processing an image, it is usually beneficial to be able to modify the key features of an image. Generally, there exist two types of image processing: Point processing and neighborhood processing. For point processing, during an image processing each pixel of the input image is processed and then mapped to the output image with its exact position. This is especially useful for changing brightness and contrast.

2.6.2 Histograms

An important requirement for image processing is the computer being able to 'see' the image. While a person can see if an image is gray-scale or if the image is too bright or too dark, the case for a computer detecting, for instance, the brightness of the image, is by using the available pixels and count the pixels of different values. This is possible using a histogram, which is described as a discrete function given in equation 2.1:

$$p(k) = \frac{n_k}{n} \tag{2.1}$$

In equation 2.1, it has been assumed that the image is gray-scale and therefore using a gray-scale intensity. n is the total amount of pixels. n_k refers to the amount of pixel with a gray-scale value of k. By dividing n_k with n, the probability of a specific gray value of k appearing on the image, $p(k)$, can be found. For color images, the principle is the same, however, three histograms are needed (due to having three channels, red, green and blue).

The histogram is generally shown as a graph, where the gray-scale intensity is shown in x-axis and the amount of pixel having that specific intensity is shown on the y-axis as shown in figure 2.5. Based on the shape of the histogram, the image features such as brightness or contrast can be detected.

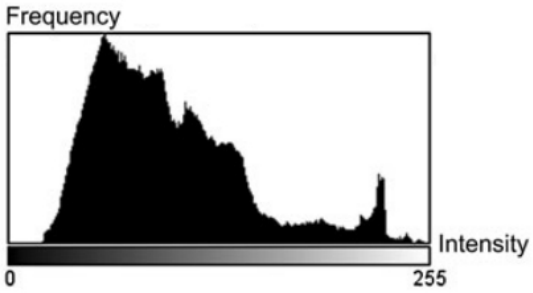


Figure 2.5. An example of a histogram. Image from [Moeslund, 2012, Chapter 4.3]

While it is visible using a histogram to detect several features of an image, it should be noticed that it is not possible to reconstruct an image based from a histogram. Thereby, it is also possible for several images to have the same histogram [Moeslund, 2012].

2.6.3 Thresholding

Another distinct feature for image processing is the ability to segment the image into several parts such as foreground with an object and background. While segmentation of an image is rather image analysis instead of image manipulation, it contains the necessary features for future image manipulation. Ideally, the segmentation between the background and object is visualized as a histogram with two significant peaks shown in figure 2.6. A histogram with those characteristics is bi-modal.

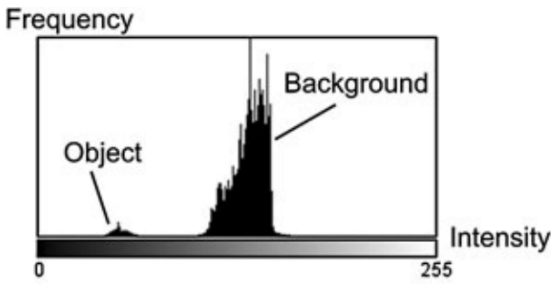


Figure 2.6. An ideal histogram [Moeslund, 2012, Chapter 4.4]

In figure 2.6, the peaks of object and background are clearly distinct from each other. However, it is generally difficult to distinguish objects from backgrounds due to the image containing a significant amount of information such as noise. One method of segmenting an image by reducing the amount of information is thresholding.

In thresholding, a threshold value T is given. If the pixel value of an input image is under or equal to the threshold value, it will be mapped to the intensity value 0 - which is shown as black. All pixel values above the threshold value will be mapped to 255 (white). The advantage of using the thresholding method is that the object and background will be distinguished as black and white respectively. For an 8-bit input image $f(x, y)$, output image $g(x, y)$ and the given threshold value T, the algorithm can be described as the

following equation 2.2:

$$\begin{aligned} \text{if } f(x,y) &\leq T \quad \text{then } g(x,y) = 0 \\ \text{if } f(x,y) &> T \quad \text{then } g(x,y) = 255 \end{aligned} \tag{2.2}$$

During the process, some of the image's information will be lost or discarded, but the lost information is generally noise or otherwise redundant [Moeslund, 2012].

An example of using thresholding is shown in figure 2.7 and 2.8.

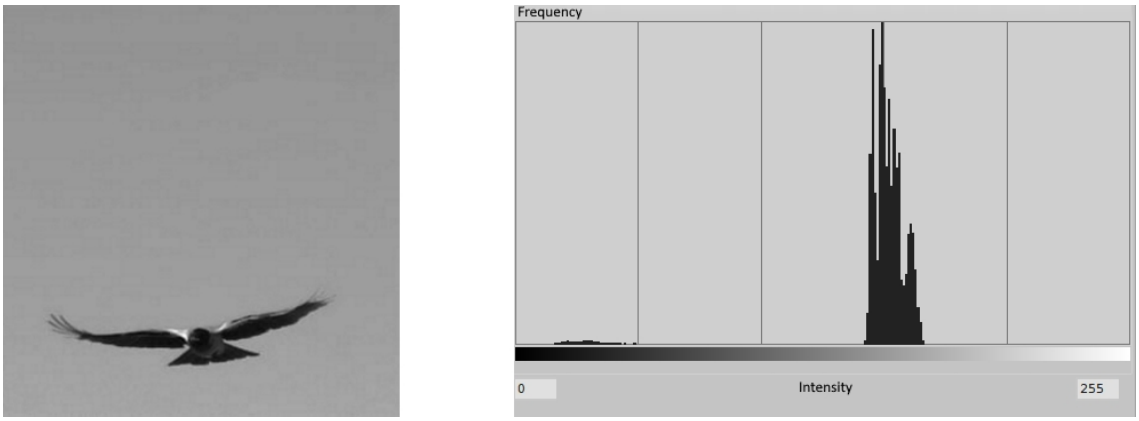
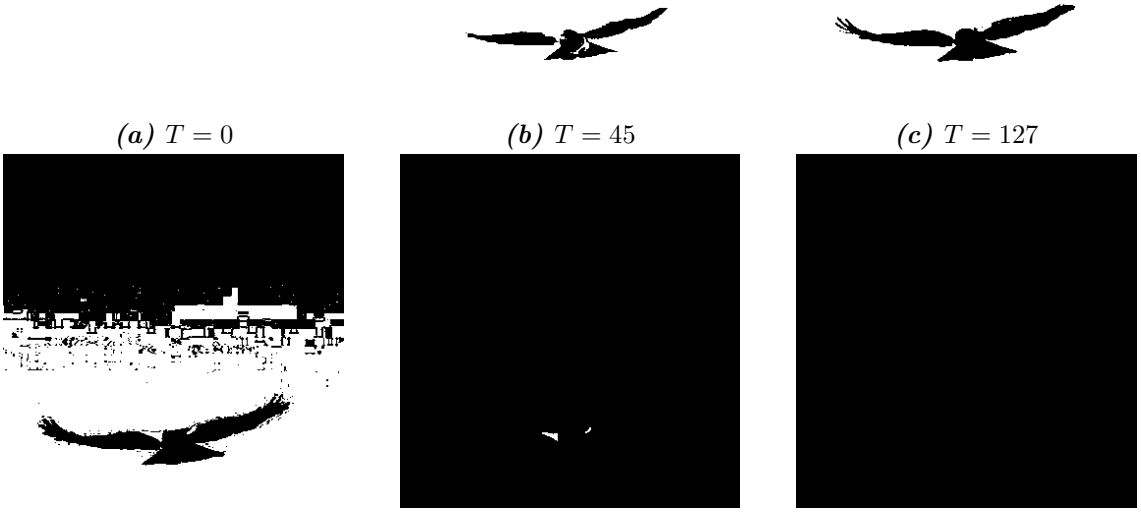


Figure 2.7. Gray-scale image with the corresponding histogram

The desired object in figure 2.7 is the bird, while the background is the sky. In the histogram, a significant peak is shown at an intensity value of around 150. This indicates the background. However, a small peak is shown at the intensity value interval of around 25 to 60. This indicates the object as previously described and shown in figure 2.6. Thereby, it is not necessary to segment the image using the thresholding method, however, it has been performed and shown in figure 2.8 for further analyzing.



(a) $T = 0$ ***(b)*** $T = 45$ ***(c)*** $T = 127$

(d) $T = 150$ ***(e)*** $T = 200$ ***(f)*** $T = 255$

Figure 2.8. Image from figure 2.7 with different threshold values

Based on the histogram shown in figure 2.7 and equation 2.2, it is seen in figure 2.8b that most of the pixels for the object have an intensity value T under 45. Due to this, those pixels are mapped to 0 and are shown in the image as black. If the threshold value is increased to 127, it is seen in figure 2.8c that there is now an almost complete silhouette of the bird. However, looking at the histogram in figure 2.7, if T is increased more, then T would be greater or equal to the intensity values of the background. This is mostly shown in figure 2.8d and 2.8e, where in the first mentioned, parts of the background pixels are now being mapped to black and in the latter, almost the entire background has been mapped to black.

It is therefore important to find a good threshold value, where it is only the object and the background that is shown and segmented. Based on the example shown in figure 2.7 and 2.8, it is seen that in figure 2.8d, the silhouette of the bird is well-defined compared to figure 2.8c, however, a significant portion of the background is mapped to black. Therefore, figure 2.8c is more suitable for image processing in that case with a clear distinction between object and background. It can then be concluded that a good threshold value generally lies in the interval between the peak of the object and the peak of the background as shown in figure 2.7.

2.6.4 Neighborhood processing

While thresholding is a great example of abstraction from the original image by only looking at one pixel for input and output as seen in the previous section, it does not say much about the pixel itself relative to the pixels surrounding it. If it was then possible to look at the neighboring pixels to get more information surrounding the pixel it would make it possible to then either remove noise or find the edges of objects [Moeslund, 2012, Chapter 5].

Starting by looking at the input window at the left side of figure 2.9 it is possible to see a 3x3 matrix of pixel values. Using the center value of the matrix to be processed the 8 neighbors surrounding it can be utilized to get an output image by sorting all the values and then finding the median as the output as seen in figure 2.9 [Moeslund, 2012, Chapter 5].

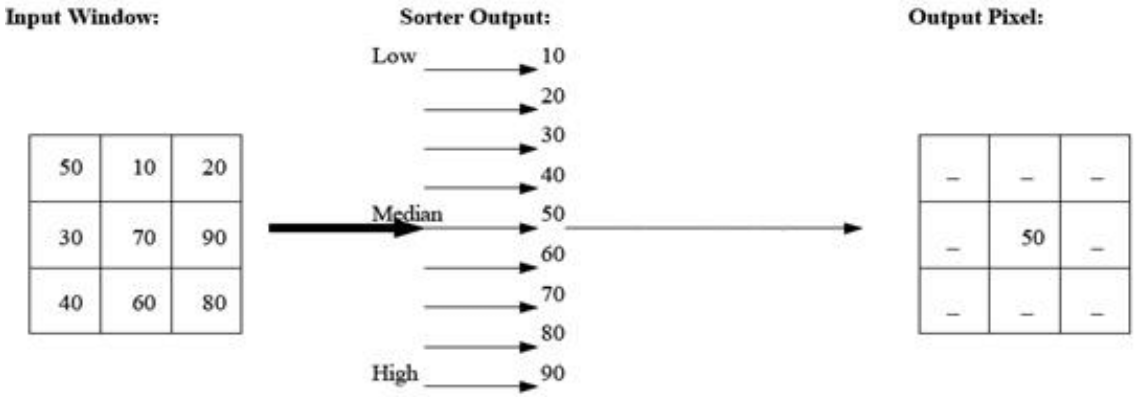


Figure 2.9. Median filter kernel and its output. Image from [Boateng et al., 2012]

This process of taking the median value of the input pixels is called a median filter. The median filter can be used to blur an image and what it is especially good at is to remove certain kinds of noise such as salt and pepper noise. Salt and pepper noise are characterized by isolated pixels having a value of 0 for black i.e. pepper or a value of 255 for white i.e. salt. This kind of noise on an image can be seen in figure 2.10a [Moeslund, 2012, Chapter 5].

(a) Salt and pepper ***(b)*** Median filter on image

Figure 2.10. Median filter removing salt and pepper noise

Using the 3x3 median filter and then going through every single pixel on the image sequentially the salt and pepper noise can be removed as seen in figure 2.10b [Moeslund, 2012, Chapter 5].

2.6.5 Morphology

Generally, the details of a captured image contain the desired object, the environment and - at times - noise. When working with computer vision, it is necessary to segment the actual important information from the redundant information, since this would make the most optimal result possible. One method of removing redundant information from the image is morphology. Morphology is about removing noise or isolating objects in binary images. Thereby, when working with morphology, it is required to work with images that have already processed with thresholding. Morphology is generally described into three abstraction levels. The basis of morphology is that the input image is compared with a customizable kernel known as the SE (Structuring Element). Each pixel of the input image is then compared with the SE, and based on the result either a Hit or Fit operation is made. A Hit operation means that if one '1' in the SE overlaps with an '1' in the input image, then the resulting output image will be '1' - and otherwise '0'. In a Fit operation, it is required that all '1's in the SE overlaps with '1's in the input image to get a resulting output image to be '1'. Hit and Fit operations are the first level of morphology [Moeslund, 2012, Chapter 6.1].

The second level of morphology builds on the previous knowledge: A Hit operation applied to an entire input image is abstractly known as **dilation**, while applying a Fit operation to the entire input image is known as **erosion**. Dilation results generally in a size increase of objects or even a merging between objects close to each other. Dilation does also fill small holes in objects, which can be regarded advantageous in cases such as identifying an object. However, due to the characteristic of dilation, redundant small objects will also increase in size. The other method erosion reduces the size of objects and is generally advantageous for isolating objects and removing noise or small objects. However, erosion might have destructive effects for desired objects: In cases of a desired object with small holes, the erosion operation might reduce the size of the object and split it into two ob-

jects. It is important to notice that the severity of the effect is depending on the size and shape of SE - a relatively small SE will not give the same effect as a larger sized one. If the SE is box-shaped, sharper edges are usually more preserved, while disc-shaped SE smooths and rounds the edges of objects [Moeslund, 2012, Chapter 6.2].

The third level of morphology is implementing dilation and erosion operations into operations which maximizes the advantages of the two operations and reduces the disadvantages. Primarily, the main used operations that use both dilation and erosion are known as **opening** and **closing**. In opening, it is desired to isolate objects but still keeping its original size. The opening operation contains of an erosion operation first and then a dilation operation of the resulting image. The motivation in using closing is to fill holes in objects but still keep the original size. Here, the closing operation contains of a dilation operation and then an erosion operation of the resulting image. It is worth noticing that opening and closing operations are idempotent, meaning that repeated operations have no further effects, if the same SE is used [Moeslund, 2012, Chapter 6.3].

2.6.6 Background subtraction

Differentiating the data desired from an image in the environment described in section 2.3 from the data that is not needed. It is then a necessity to segment the video data to make it easier to detect what is desired. The data necessary from the image is a cage, that is used to distribute packages. It is then possible to make the assumption that this cage that needs to be detected is in motion in the video sequence, which needs to be differentiated from things that do not move in the video sequence. Here it is possible to use background subtraction, where the apparent background i.e. a reference image is removed in image processing which only leaves things that are in motion where then it is possible to further process it with i.e. thresholding. This method is simple and very efficient to make it possible for finding objects in a video sequence if it is in a controlled environment where the apparent background does not change in any capacity [Moeslund, 2012, Chapter 8.3].

Though this apparentness can still be a false conclusion, because even if it is indoors there will be windows and doors that can influence the scene with outside light such as sunlight. Because of this reason a static background might not be possible, and if it violated significantly it might give incorrect results. In these cases it might be appropriate to apply image differencing instead, which operates as background subtraction with the exception of not using a reference image as a background but the previous image instead. Image differencing is therefore a great method to measure the changes in an image, but with this there is a problem not faced by background subtraction. This problem is that there is no possibility to detect new objects that are not in motion. Meaning that if a new object came into the image and then became stationary, it will not show up in the next frame, which is a clear and big down side compared to background subtraction that only cares about if the object is different from the reference image [Moeslund, 2012, Chapter 8.4].

2.7 Problem Statement

Based on the analysis described in this chapter, following problem statement has been formed:

How can computer vision be implemented to track and determine the directionality of a cage with packages?

State of the art 3

3.1 Image Processing Techniques

In this section, the basics of image and image manipulation techniques are described.

3.1.1 Histogram

Histograms have so far been described in section 2.6.1 as a tool to recognize distinct features of an image, but they can also be processed such that it can be used as an image manipulation tool - a term known as histogram processing. There exist different types of histogram processing regarding the type of operation needed.

Histogram stretching

If the contrast is too low, the image might appear gray and unclear for humans. The characteristic of an image with low contrast is a narrow difference of gray-level intensity which is shown in the corresponding histogram of the image. An example of a gray-scale image with low contrast is shown in figure 3.1.

A solution for this problem is to map the gray-level values into an other value such that the level of gray-level intensities are spread out, which improves the contrast. This method is known as histogram stretching. Based on the original image shown in figure 3.1, histogram stretching has been used to improve the contrast as shown in figure 3.2.

Figure 3.1. Low contrast image with its corresponding histogram

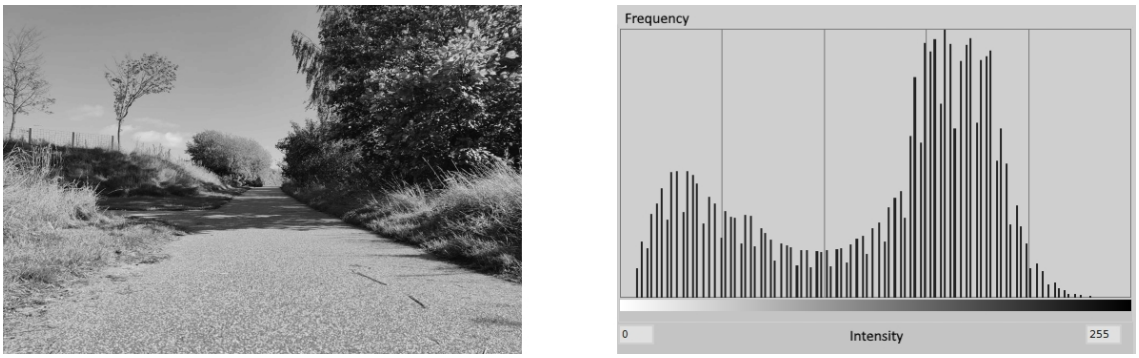


Figure 3.2. Improved contrast image from figure 3.1 with its corresponding histogram

While the contrast has been improved in figure 3.2, it is shown in the histogram that not all intensity values are used.

Histogram Equalization

An alternative method of histogram manipulation is histogram equalization. The motivation of histogram equalization is to distribute the frequently used intensity values evenly across the entire intensity range unlike histogram stretching, where the frequency of a given intensity value remains constant. This is done by using a cumulative histogram instead of a standard histogram. The resulting image using histogram equalization generally increases the global contrast of the image. A comparison between histogram stretching and histogram equalization is shown in figure 3.3:

(a) Input image

(b) Histogram Stretching

(c) Histogram Equalization

Figure 3.3. Comparison between histogram stretching and histogram equalization with their corresponding input image

Comparing histogram stretching and histogram equalization in figure 3.3b and 3.3c respectively, it can be seen that the contrast has been notoriously increased in the entire image by using histogram equalization.

This method generally works best when the histogram of the image is confined to a particular region of an image. If the image contains large intensity variations where histogram covers a large region, undesirable results such as over-brightness can happen [Sudhakar, 2017].

CLAHE (Contrast Limited Adaptive Histogram Equalization)

CLAHE is a variant of AHE (Adaptive Histogram Equalization)), which is an image processing technique used to improve the contrast in images. AHE works by dividing the image into several small blocks called *tiles*. Then a histogram is computed for each tile of the image. The multiple histograms are then used to redistribute the lightness values of the image. AHE is therefore very effective improving the contrast and enhancing edges in the image, but it has also a tendency to over-amplify noise in images. In CLAHE, the contrast amplification is limited by using a specified maximum contrast value. If any histogram bin is above the maximum contrast value, those pixels will be clipped and distributed uniformly to other bins prior to applying histogram equalization [Sudhakar, 2017]. An example of using CLAHE on a low contrast image is shown in figure 3.4, where the input image is the same image shown in figure 3.3a.

(a) Histogram Equalization

(b) CLAHE

Figure 3.4. Comparison between histogram equalization and CLAHE

It is shown in figure 3.4 that when comparing the image processed with histogram equalization and CLAHE, while the contrast is noticeably less than in histogram equalization, the resulting image in figure 3.4b improves the visibility level of the input image.

3.1.2 Neighborhood processing

Bilateral filter

One of the filters using neighborhood processing is the bilateral filter. This filter smooths the image and is noise-reducing and edge preserving. This has been done by replacing each pixel value with weighted average of nearby pixel values. To preserve the edges, the intensity variation is also taken to account. Thereby, the filter acts lightly on regions with a high variety of color, since these regions are expected to be edges. In regions with uniform colors, the filter acts strongly by removing the fine details - smoothing the image. The filter is therefore an effective way to smooth image while preserving the edges in the image [Paris et al., 2008]. An example is showcased in figure 3.5 with the input image being a gray-scale image of figure 2.3 from section 2.3.1. Here, the image processed with bilateral filter is compared with median blur which was described in section 2.6.1 .

(a) Bilateral filter ***(b)*** Median blur

Figure 3.5. Comparison between bilateral filter and median blur

Comparing figure 3.5a with 3.5b, it can be seen that the image with bilateral filter seems to be able to have kept the edges despite the image being blurred. Bilateral filter is therefore an efficient way to remove fine details, which from an object detection perspective is assumed to be redundant information, but still keep the necessary edges.

Gaussian blur

A Gaussian blur is a commonly used filter for reducing noise and unnecessary information. Like many others, it utilizes neighborhood processing, however it's key characteristic is the weighted kernel utilized. The kernel weights adjacent pixels higher than those further from the center. This falloff is often approximated as an exponential decay, which can be described as follows

$$\frac{2^{m-1}}{2^d} \text{ where } d = |\Delta x| + |\Delta y| \tag{3.1}$$

It is worth noting that formally, a Gaussian blur would utilize the Gaussian distribution as opposed to this approximation. As seen in figure 3.6 the visual comparison of the approximated and a more accurate Gaussian blur kernel can be seen:

1	2	4	2	1
2	4	8	4	2
4	8	16	8	4
2	4	8	4	2
1	2	4	2	1

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Figure 3.6.
Right: Exponential decay 5x5 kernel
Left: Rounded Gaussian distribution 5x5 kernel

The main benefits of utilizing a Gaussian blur is to remove noise, while preserving edges. While the edges are not as clean as they would have been, had a bilateral filter been utilized, it runs substantially faster, often making it ideal for applications that are time sensitive.

Edge detection

Edge detection is the process of finding the boundary of an object, meaning that in almost all cases, edges will represent the object in the image. The edge of an object can then be used to get a higher level of abstraction of the image, which means that the information in the image will be less thus easier to process. Before it is possible to find the edge of an image it is important to define what it is. Using a gray-scale image as a baseline it is possible to define an edge as a position where a significant change in gray-level takes place. Looking at figure 3.7 on the left there is shown an image. To find an edge on that image a slice of that image can be taken vertically that is one pixel wide, which is illustrated in the figure between the arrows. By using this slice, a graph can be made where we can interpret the intensity value in the slice as the height in the graph that is shown on the right of figure 3.7. When looking at the graph when there is a significant change in the gray-scale value in the slice, the graph will also have a significant change in the height. These significant changes in the height is illustrated as circles in the graph, and is where the edges is defined in an image [Moeslund, 2012, Chapter 5.2.2].

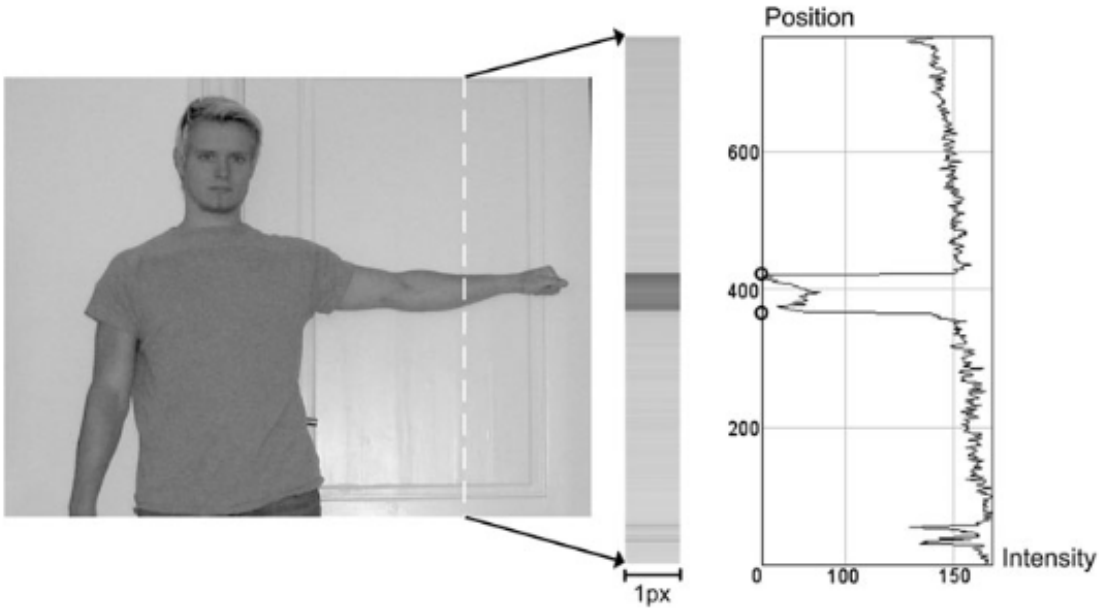


Figure 3.7. A single column of the image is enlarged and presented in a graph. This graph contains two very significant changes in height, the position of which is marked with circles on the graph. This is how edges are defined in an image. Image from [Moeslund, 2012, Chapter 5.2.2]

To detect edges in an image the directional change in the intensity can be used, this concept is gradient in image processing. This can be done by representing the previous image in a 3D graph as seen in figure 3.8. In the graph it is shown that the width and height of the image is represented by the x- and y-direction respectively, whereas the z-direction is represented by the intensity of the image interpreting it as height.

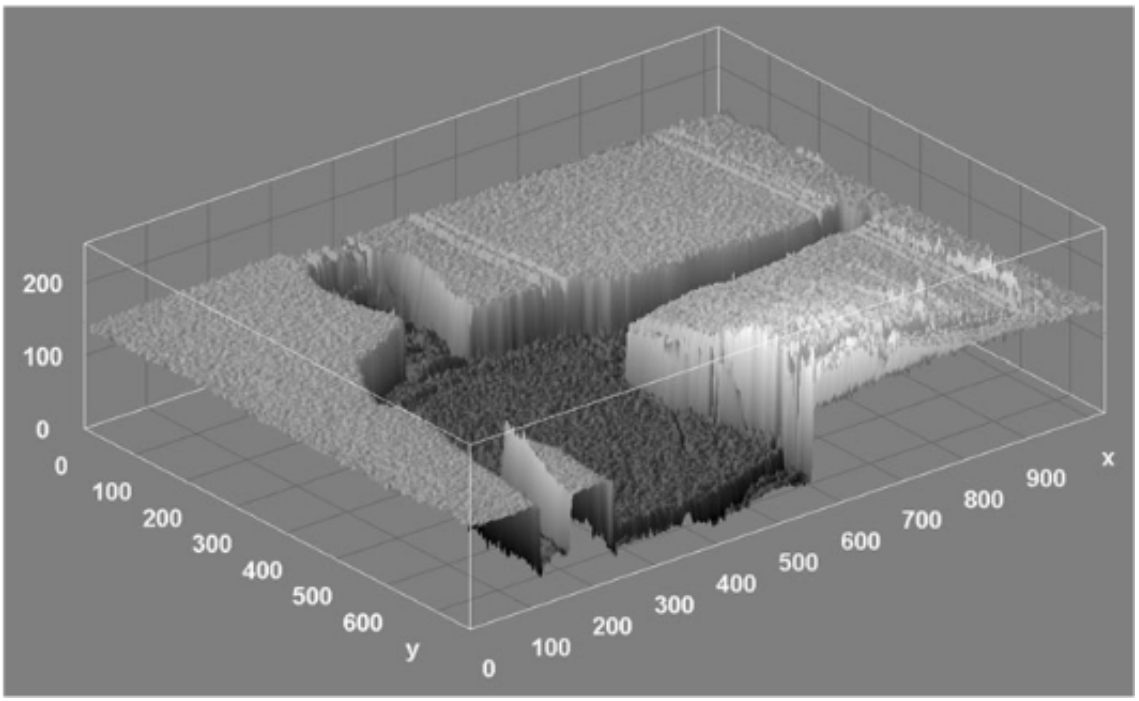


Figure 3.8. A 3D representation of the image from figure 3.7, where the intensity of each pixel is interpreted as a height. Image from [Moeslund, 2012, Chapter 5.2.2]

For each point in the image there would be two partial gradients that would span a plane in the x- and y-directions intersecting a chosen point. These partial gradients can be used to determine the direction of the edge, as well as using their derivatives to calculate the resulting gradient by using eq. 3.2

$$\overrightarrow{G}(g_x, g_y) = \begin{pmatrix} g_x \\ g_y \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} \tag{3.2}$$

A gradient also has a magnitude, which is how steep the change of height is in the direction of the gradient and is also the length of the gradient vector. The length of the gradient vector can be calculated using the Pythagorean theorem solved for magnitude as seen in eq. 3.3

$$Magnitude^2 = g_x^2 + g_y^2 \Leftrightarrow Magnitude = \sqrt{g_x^2 + g_y^2} \tag{3.3}$$

For faster implementation the magnitude can be approximated by using eq. 3.4

$$Magnitude = |g_x| + |g_y| \tag{3.4}$$

As the partial gradients are first order derivatives it can not be calculated because an image is not a continuous curve and therefore needs an approximation. To calculate the approximation of the gradient it is possible to use the difference between the previous and next value seen in eq. 3.5

$$g_x(x, y) \approx f(x + 1, y) - f(x - 1, y) \tag{3.5a}$$
$$g_y(x, y) \approx f(x, y + 1) - f(x, y - 1) \tag{3.5b}$$

This approximation will result in a gradient value that is positive when the pixel change from dark to bright and negative when it is reverse. It is possible to get the opposite value by switching the signs seen in eq. 3.6

$$g_x(x, y) \approx f(x - 1, y) - f(x + 1, y) \tag{3.6a}$$
$$g_y(x, y) \approx f(x, y - 1) - f(x, y + 1) \tag{3.6b}$$

Equation 3.5 can be applied with a 2D kernel using correlation, a 2D kernel is used because a 1 dimensional kernel is too sensitive to noise. A popular 2D kernel that is used is the Sobel kernel where the center pixel is weighed more which can be seen in figure 3.9

Sobel

Vertical		
-1	0	1
-2	0	2
-1	0	1

Horizontal		
-1	-2	-1
0	0	0
1	2	1

Figure 3.9. A horizontal and vertical Sobel kernel. Image from [Moeslund, 2012, Chapter 5.2.2]

What happens when using the Sobel kernel is that the coefficients multiply the corresponding values around the source pixel then adding them together to get a sum of the values. This value will be a negative number if it goes from a bright to dark or in other words a bigger number to smaller number, in the other spectrum if it goes from a smaller number to a bigger number it will be a positive number. In the cases where the pixels are uniform the number given will be 0. This process where you calculate the sum can be seen in figure 3.10

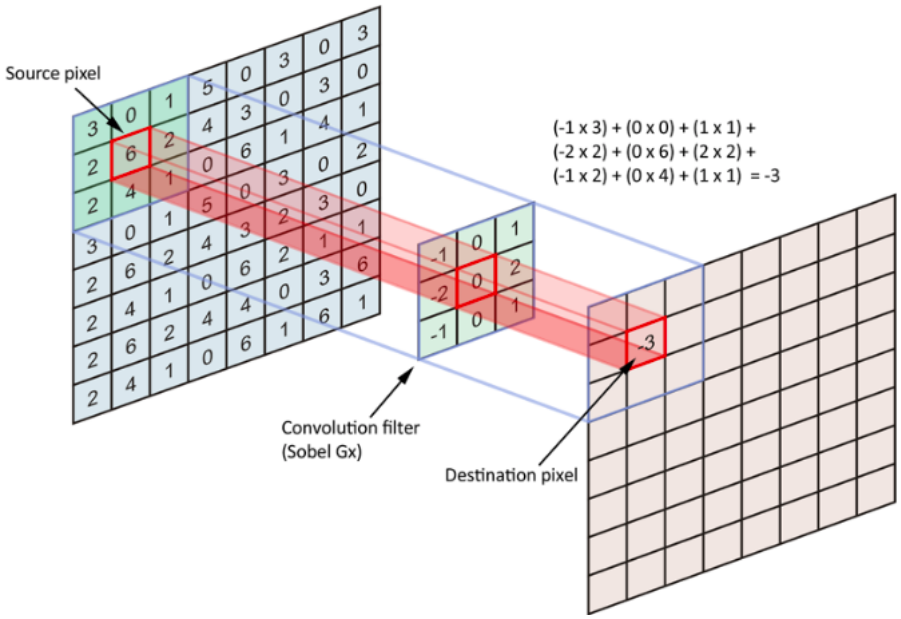
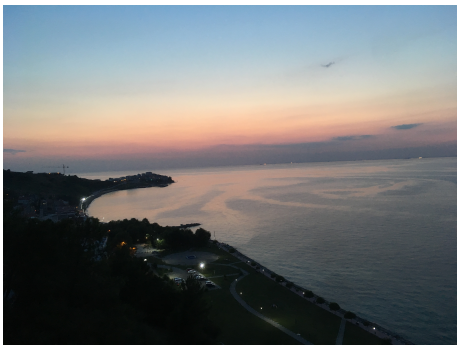


Figure 3.10. Sobel kernel calculation of convolution. Image from [Du, 2020]

It is possible to use the Sobel kernel by combining the vertical and horizontal correlated images and then binarize the resulting image by thresholding it, but a better solution is to use the canny edge detector. The canny edge detector is an algorithm that uses the Sobel kernel and also gives pixel-thin edges, this process is done in 5 steps:

1. Apply a Gaussian filter to smooth the image and remove noise
2. Find the gradients using the Sobel kernel
3. Apply non-maximal suppression, by comparing a pixel with its neighbors in the gradient direction
4. Apply double threshold comprising of T_{high} and T_{low}
5. Categorize an edge that is localized higher than T_{high} as a strong edge and as weak edge if between T_{high} and T_{low}, then only keep edges that are strong or connected to a strong edge

The difference between only using the sobel kernel for edge detection and using canny edge detection can be seen in figure 3.11



(a) Original

(b) Sobel

(c) Canny

Figure 3.11. Image processing with different methods of edge detection

3.1.3 Color masking

While thresholding have been described in section 2.6.1, it has been described for a gray-scale image. Thereby, an image is generally converted from color to black-and-white, and then the thresholding process is used. However, it might be the case that it is desired to threshold one specific color or a range of colors. While it is possible to use the RGB color space to select the desired color and segment it from the rest of image, a problem is that this solution is preferable to a fixed image but not to a video recording or streaming, where disturbances such as lightness, which was described in section 2.3, can give the object an entirely different intensity of color and thereby inconsistent results. One method of solving this problem is by separating the color and intensity. The HSI color representation is one example of using this method. The color is now being represented by hue (the pure color), saturation (how much the color has been mixed with white light) and intensity (the average of red, green and blue color components), hence it's abbreviation. The separation of color information and intensity means that the system becomes more robust towards noise or lighting changes. In computer vision, HSV (hue, saturation, value) is commonly used instead of HSI, but the difference between these are relatively minor: Value in HSV refers to the intensity of color correlated from the range of dark to the pure color or hue [Mamdouch, 2020].

Requirement specification 4

4.1 Scenario

To solve the problem statement described in chapter 2, the system needs a means to detect a cage, to identify the cage, to observe the movement of said cage, to analyze these observations, and to store the resulting information. To detect, and identify the cage an RFID sensor is a functional solution. It will register the cage and its associated ID when the cage is within range. To observe it, a camera will be utilized. This will stream the movements of the cage to the processing device. To process this data, a PC or a specialized device is required to do the heavy lifting. A database will be used to store the results of the analysis, along with the RFID tag that is associated.

1. The system can start only when the RFID sensor has detected the cage is in range and then get the cages ID, which can the be stored
2. After the cage has been detected the camera can start streaming the video and the movement of the cage.
3. With the streaming data the footage can be processed by a computer in real or near real time to identify the cage and detect the direction of the cage.
4. When the direction of the cage has been found it can then be sent, along with the cages ID, to a database which then can be stored for easy access and readability.

From the presented scenario, an overview of the systems functionalities can be outlined. This is done by a use case diagram where the actors are the PC (computer), the RFID sensor, the camera, and the database where the primary actor is the PC. The use case diagram is shown in figure 4.1

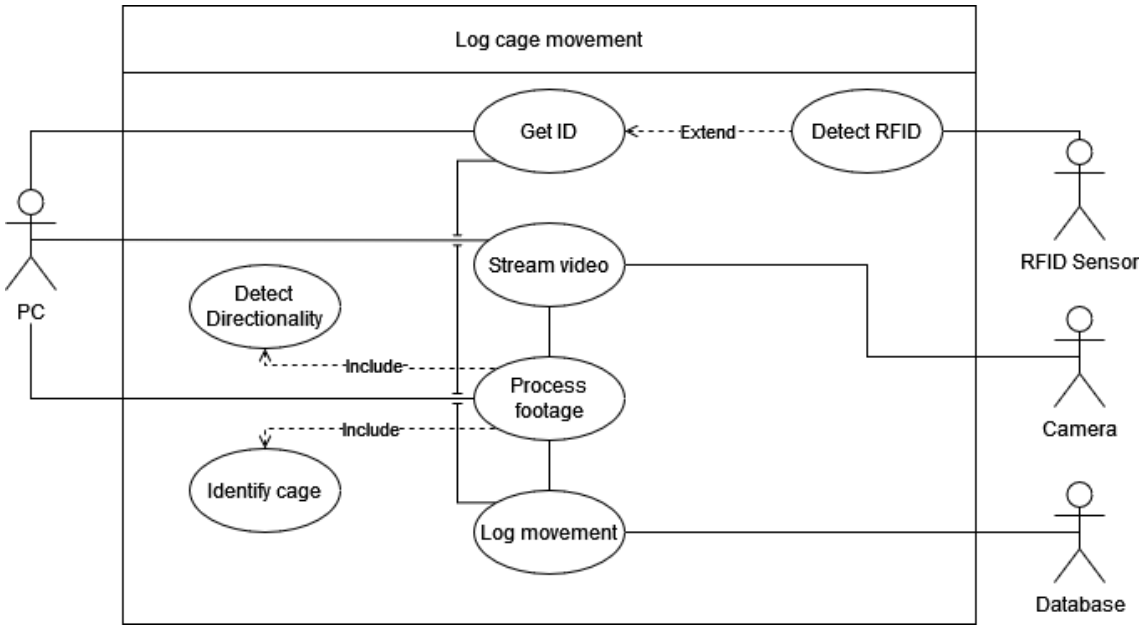


Figure 4.1. A use case diagram showcasing the initial thought process on how to make the prototype

Lyngsoe has suggested an alternative approach which would be more functionally viable. In this approach, the camera will stream video footage, to the processing device, which will identify a cage if one is present. It will then request an ID from the RFID sensor, while simultaneously attempting to identify directionality of the cage. The results should then be stored in a database for further use.

1. The system can start only when the camera has detected the cage within it's field of view.
2. After the cage has been detected the RFID sensor can attempt to obtain the ID which will be stored
3. With the streaming data the footage can be processed by a computer to detect the directionality of the cage.
4. When the direction of the cage has been found it can then be sent, along with the cages ID, to a database which then can be stored for easy access and readability.

Based on this alternative, another use case diagram will be generated. It has the same actors as the initial use case and can be seen in figure 4.2

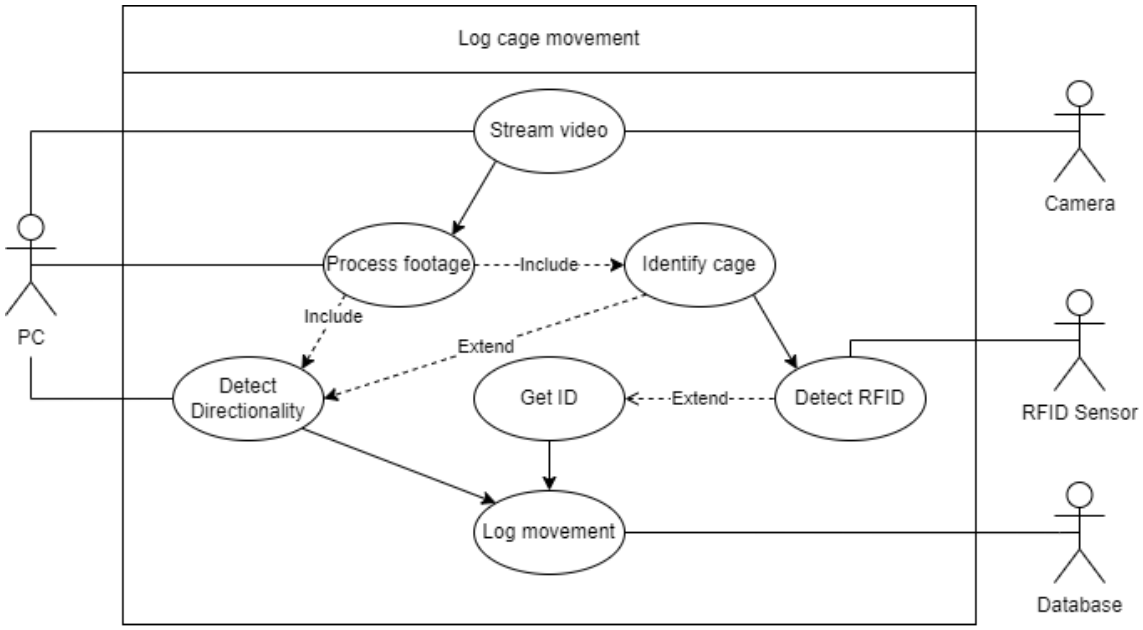


Figure 4.2. An alternative use case given from Lyngsoe systems

The RFID element of this use case is not within the scope of this project, and will be omitted henceforth, as agreed upon with Lyngsoe Systems. As such, the use case diagram looks as following

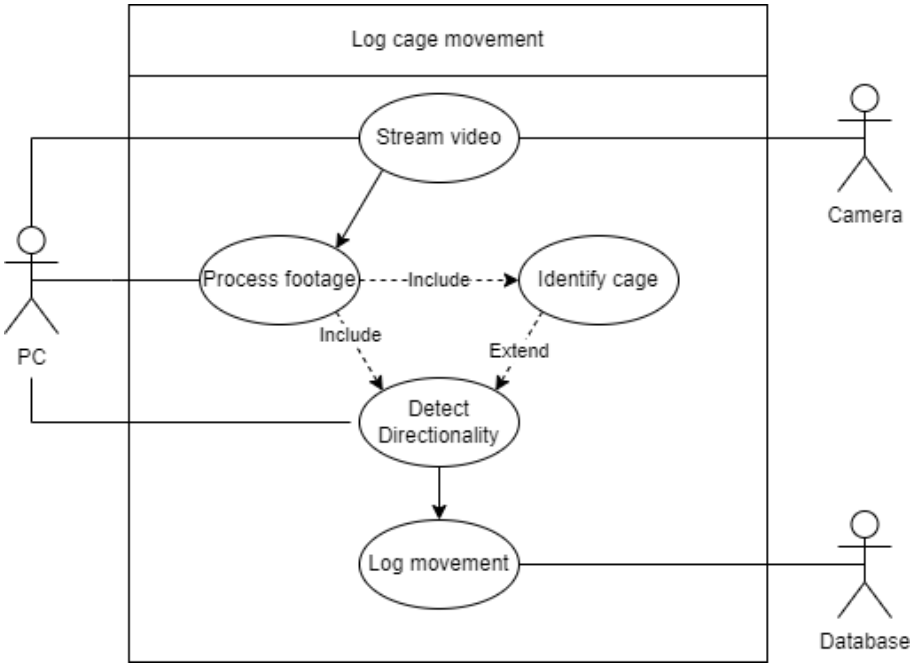


Figure 4.3. The final use case that will be used

4.2 Requirements

Based upon the problem analysis in chapter 2 and state of the art research in chapter 3 combined with the use case shown in figure 4.3 it is possible to outline the requirements for a functional system.

4.2.1 Functional requirements

1. The camera must always be running
2. The image resolution must be set to 1280x720 pixels
3. The camera must have a maximum frame rate at 15 fps
4. The computer must be able to receive data from the camera
5. The computer must be able to process the camera data
6. The computer must be able to object-detect the cage
7. The computer must detect and store the direction of the cage

4.2.2 Non-functional requirements

1. The camera lens must not be distorting
2. The used camera must be mounted to be facing vertical
3. The computer must be able to access and process the camera data before the next cage
    - If there is more than a single cage at once then it should be able to differentiate the cages direction
4. The system should have easy integration for RFID

Design 5

Based on the requirement specification described in chapter 4 and the final use case shown in figure 4.3, a prototype can be developed. This chapter contains the choices made and present what is needed to develop the prototype.

5.1 OpenCV

Based on the requirement specification in chapter 4 and the use case to the prototype shown in figure 4.3, a software library to process and manipulate images are needed.

OpenCV is an open source computer vision library known to be the de facto standard tool for real-time image processing and computer vision. The library is thereby built for maximum efficiency and performance of computing intensive computer vision tasks and is therefore faster than the other competing computer vision tools such as MATLAB [Srivastava, 2020]. Inside computer vision, the OpenCV library contains over 2500 algorithms with extensive documentation and thereby has a large application area spanning from basic elements such as simple image manipulation to object recognition or machine learning [Boesch, 2022].

It has been concluded that OpenCV is capable to be utilized to make the prototype. The fast performance of the library compared to other competing libraries/frameworks means that it would be advantageous to use OpenCV for the project. Thereby, it is concluded that OpenCV will be used to develop the prototype.

5.2 Determination of directionality of the cage

To consistently determine the directionality that the cage moves, one approach is to compare the pixel coordinates at which the cage has been observed. From these comparisons a directionality can be determined. This basic solution grows more complicated as edge cases and common errors are introduced. One of these complications is a cutoff. Where does the movement of one cage end, and the next begin. The decided solution is a mixture between a watchdog timer, and a cutoff based on the coordinates of the cage. A watchdog timer is a timer normally utilized to restart frozen or softlocked software, to keep it running. This feature is repurposed to identify when an object is no longer seen. If a cage has been detected within the cutoff region, or has not been detected for a certain number of frames, it is assumed to have left. The cutoff region is a square region at the edges of the individual frame, which can be used to be checked against to know when a cage is leaving the frame. The advantage to taking this approach, as opposed to others is that it covers most edge cases caused by this step. While the cutoff region

is the main approach, the watchdog is supplementary, and will serve primarily to handle exceptions. It should be noted that a watchdog timer does not need to be very long to provide a high level of certainty, as the odds of repeatedly failing is an exponential decay as seen in equation 5.1.

$$(1 - accuracy)^{samples} \tag{5.1}$$

Assuming an accuracy of finding the object is 90% that means just 3 frames would make the odds of failure 0,1%. It should be noted that while this is correct statistically, in practice, the failure may have been caused by the physical environment, which is likely to persist across multiple frames. Another complication is the approach to calculating the movement. It is possible to make vectors between the pixel coordinates at which the cage has been observed from one frame to another. If the vectors between the individual frames are added up, a misreading of the last frame would be all that is needed to cause a failure. If the directionality of the individual vectors from frame to frame are counted as *votes* for either the cage going left or right with majority vote being the final output, then stopping up midway for a couple of seconds would introduce substantial votes generated purely from noise. This approach would not work either. An alteration of the voting system will be our final approach to this part. It will compare the points to the next one, offset by half of the list, as demonstrated in figure 5.1

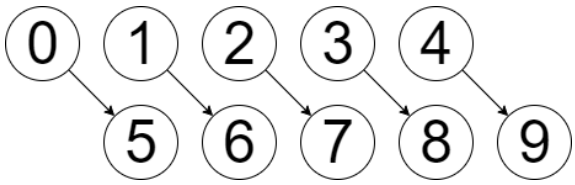


Figure 5.1. The relation between the logged pixel coordinates from frame to frame, numbered chronologically, and the movement vectors generated from these

These vectors will be tallied as votes for directionality, and will determine the final conclusion. By taking this approach, as much as half of the frames could be stationary without any noise being introduced by idling in frame, while eliminating the single point of failure introduced by summing the vectors. this offset also eliminates many edge cases for slow moving cages.

5.3 Contour

Contours can simply be described as a curve that joins all continuous points along the boundary. In OpenCV, finding contours are based on finding the boundary of objects in a binary image. This means that prior to finding contours of objects, it is important to apply processes such as thresholding and morphology to get a binary image as described in section 2.6 and chapter 3 [OpenCV, 2021].

The motivation of using contours is that the data is stored as a short list of boundary point coordinates, which take up very little data, as opposed to drawing it on the frame, which is much more data. Each individual contour stores the boundary point of the object in x- and y-coordinates together with the width and height of the object [OpenCV, 2021]. While the contours stores several objects, it might be that many of those objects are either

redundant or artifacts. By defining the range of possible width and height, the contours can be limited to only the desired contour for the object. Thereby, a contour for the desired object is found and stored.

There exist several feature types that relates to a cage as described in section 2.3.1, but regarding contours the primary part is the area and shape. Since cages commonly are rectangular-shaped, the contour should generally have a shape of a rectangle. In case of the object in the binary image not having a rectangular shape due to noise such as lightness as described in section 2.3.1 and 2.5, it is then possible to reconstruct and plot a rectangular contour on the object by using the given coordinate sets. The edge of the coordinate sets are then used to fit the rectangular contour around the object.

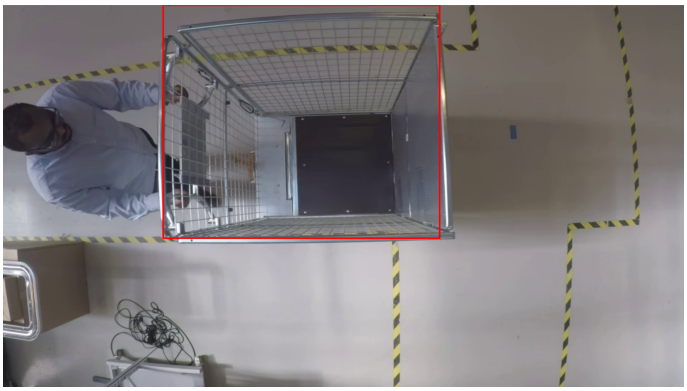
The key priority for contours is to find the center of mass of the rectangular contour. The motivation of finding the center of the contour is that this is the center of the object. This can be used further to determine the directionality of the cage as described in section 5.2.

5.4 Segmentation of data

As described in the previous section it is possible to find the cage using contours by thresholding and morphology. To make these processes as accurate as possible there needs to be a segmentation of the data, so that it is only the cage that is in the ROI (Region Of Interest).

The data can first be segmented by finding what is in motion and what is not in motion, and this can either be done by background subtraction or image differencing as described in section 2.6.6. It is also possible to determine that background subtraction is possible because the lighting conditions in a sorting center have constant artificial lighting with no natural light as described in section 2.5. In the standard OpenCV python library i.e. opencv-python, there are only 2 different background subtraction methods, these being MOG2 (Mixture-of-Gaussians 2) and KNN (K-Nearest Neighbors). There is not many differences between the outcome of these methods, and it is possible to use both to background subtract.

When the foreground has been found and only things that are in motion is left, there would then be a need to find the cage in the image and not other possible objects such as a person. To make this possible we can try to make a mask for the cage that can then be used to subtract other possible objects from the image. This can be done by color masking the image against the color of the cage described in section 2.3.1, however this still makes it possible to mask other things with the color of gray. So to make it more reliable at finding the cage, edge detection can be used as another mask and then make an and operation between the two masks that only leaves things that are in both images. Doing this will give a much better mask for the cage, but will leave edges of other object that is not the cage. Using morphology on the composite mask it is possible to isolate the cage. Since the problem that needs to be solved is for cages that have packages, defined in the problem statement in section 2.7 it is also imperative to find the packages inside the cage. Therefore, the isolated cage can be used as a ROI which then can be used as an image. This is illustrated in figure 5.2.



(a) One frame where it is possible to see the cage

(b) The ROI after isolating the cage

Figure 5.2. The ROI extracted from the full image

After getting the ROI color masking can be used to find the packages, since nearly all packages have a brown color with maybe colored tape. This color masking can then be used to find a position in the cage that can be compared to the next position to find the direction.

5.5 System Flow

The system flow can be described as a flowchart. For the sake of readability, some of the flowchart has been reduced to predefined functions, which have each gotten their own flowchart, as seen in figure 5.3 and 5.4, with the main flow chart shown in figure 5.5

In figure 5.3, Data manipulation #1 is shown, which outlines the procedure for which the cage will be isolated from the rest of the image, and its contours will be found. Data manipulation #2 is also shown in the figure, which outlines the procedure for isolating the package inside the cage, and finding its contours. In essence this is mostly the same procedure as the one above, however, since color plays a relevant role, it has a few additional steps.

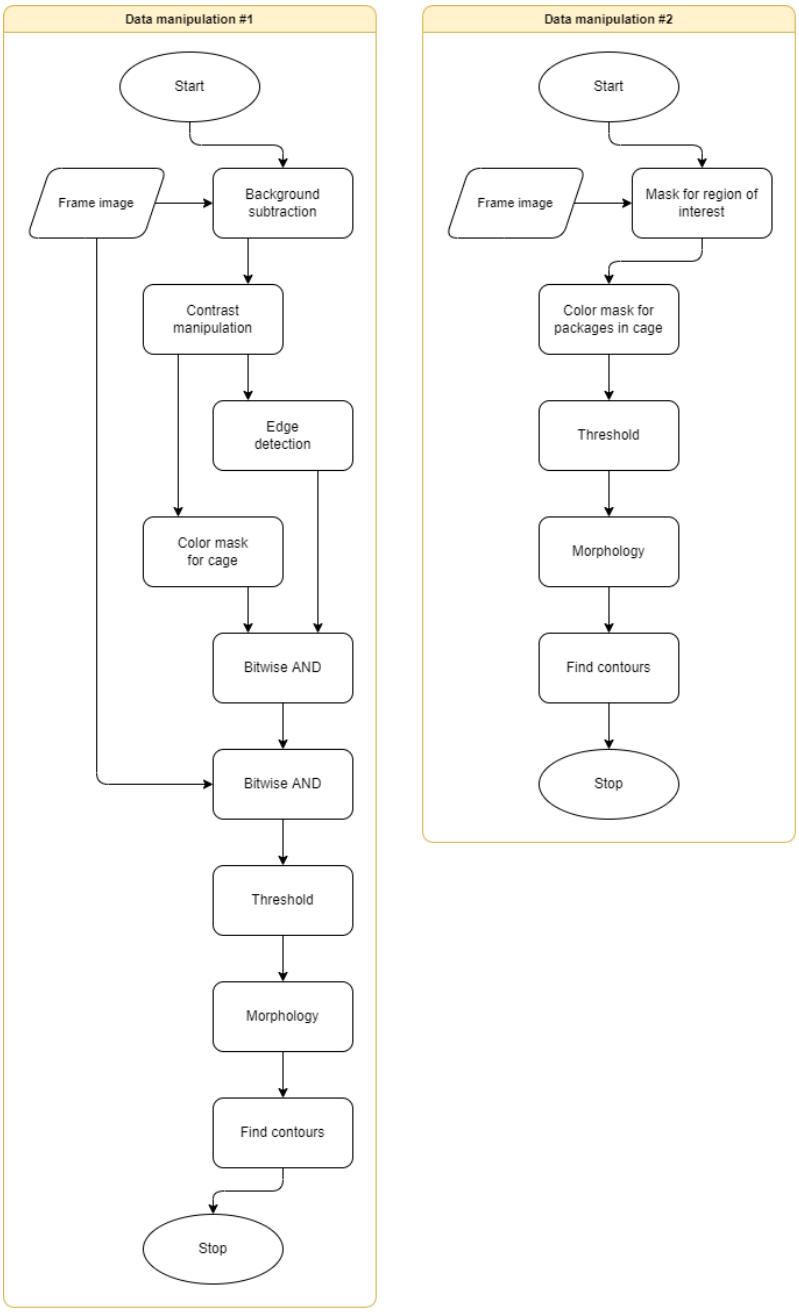


Figure 5.3. 2 flowchart showcasing data manipulations of a frame

In figure 5.4, another set of predetermined functions, position estimation and termination check is found. Position estimation is the procedure by which a point is estimated, in case the cage is not spotted for a frame. Termination check is the procedure which checks whether the cage has been seen in the cutoff region, or the watchdog timer has run out. It then also calculates the final output based on the positions logged by the system.

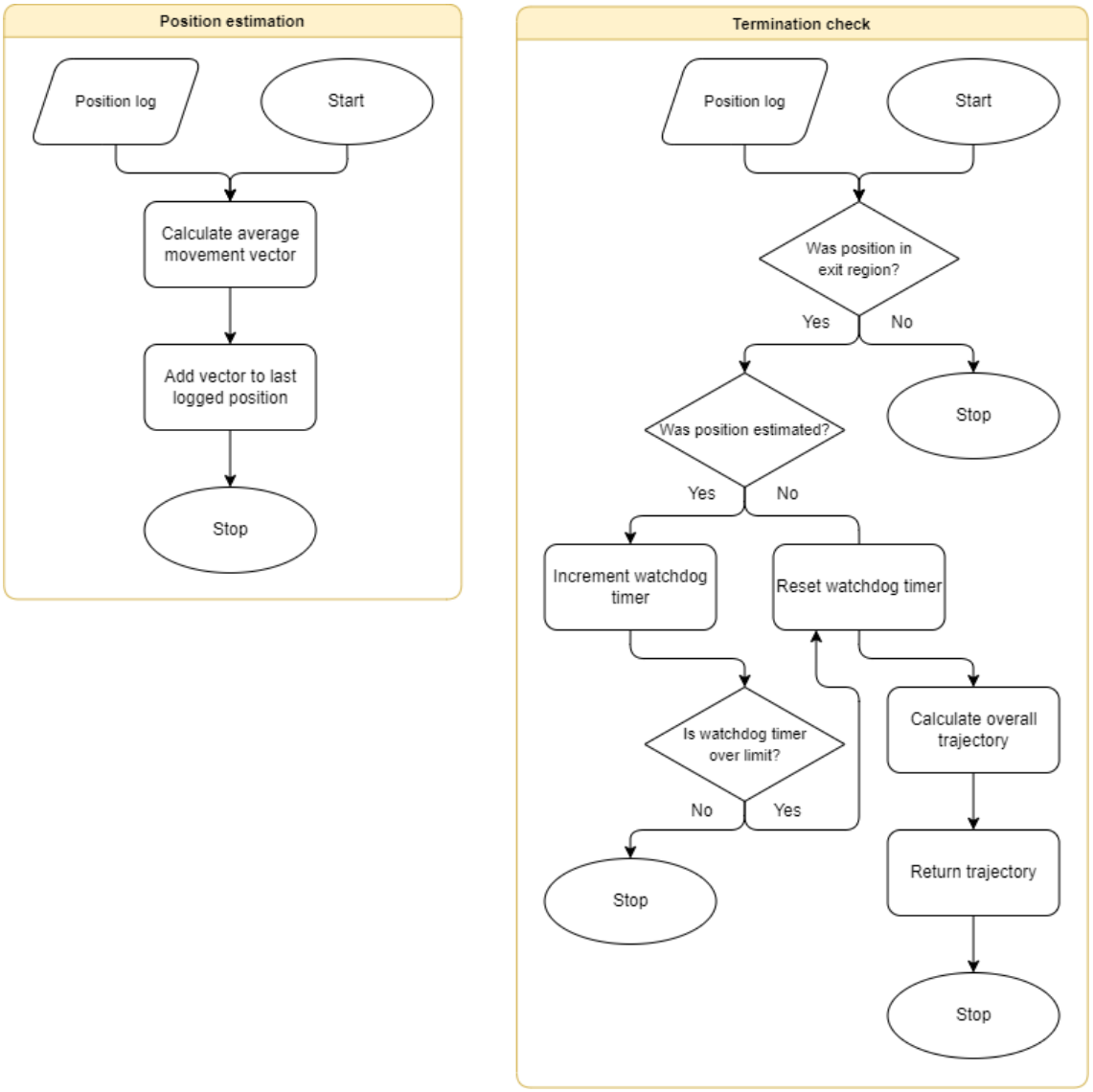


Figure 5.4. Flowchart for the termination and estimation processes

In figure 5.5 the main system flow is illustrated, split in two. The system tries to find a box with contours, and tries too find a package in the box, which it will then find the position of, and log. If it fails at this at any point in the procedure, it will attempt to estimate the position and log that instead. After every log, it will check if the termination condition is met, and if so, it will output the calculated direction, and end the process.

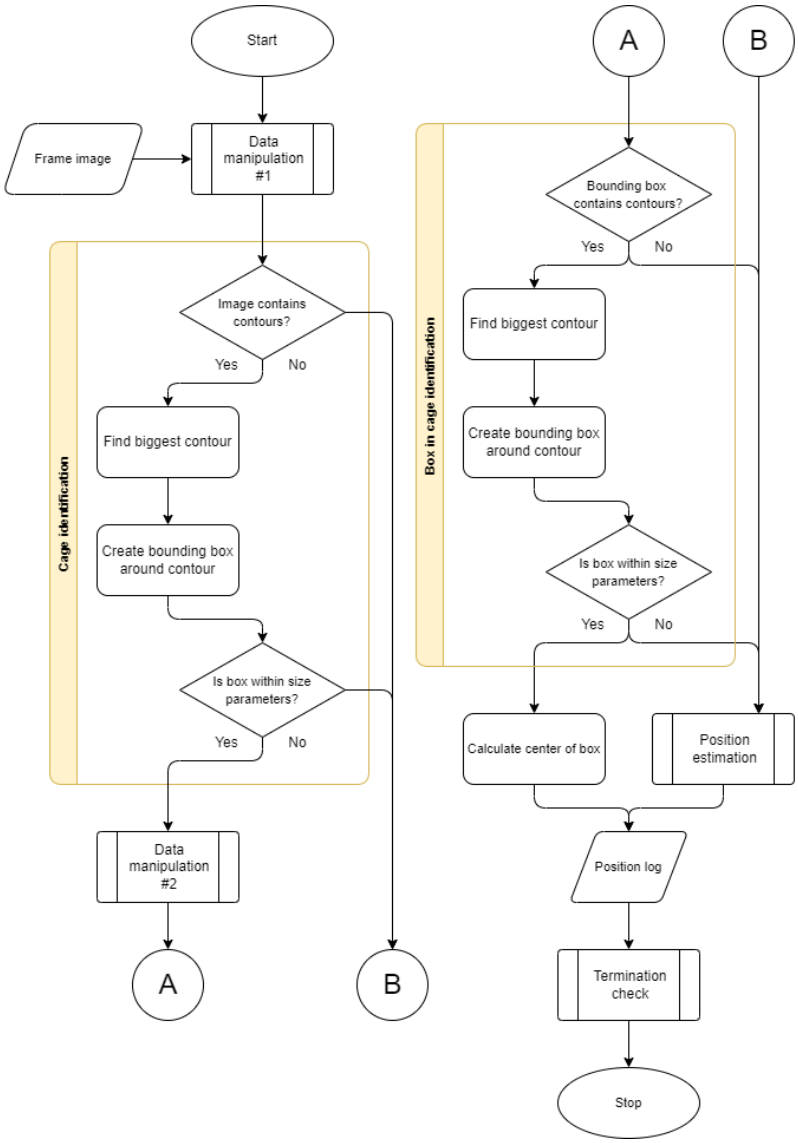


Figure 5.5. The general flow of the system

Implementation 6

Based on the design segment in chapter 5, a prototype has been developed to demonstrate the feasibility of the system. This chapter will outline the implementation of said prototype, and its features. The camera used for the implementation of the prototype is a GoPro Hero 5, primary chosen for its accessibility. The image resolution and frame rate have been fixed to 1280x720 and 15 FPS respectively to comply with requirement 2 and 3 in functional requirements described in section 4.2.1. The full code can be found on Github.

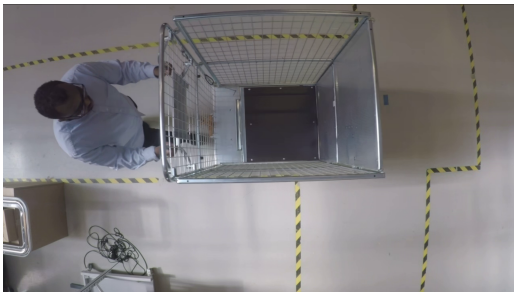
6.1 Background subtraction

The code described in listing 6.1 first creates the rudimentary background subtraction, which was described in section 2.6.6. The output of this is then put through several iterations of morphology closing and openings to remove noise. The not-isolated Region Of Interest is applied to the original frame, using `BitwiseAND`, which masks out all irrelevant information. it is then normalized, and CLAHE is used to smooth the lighting and highlight edges.

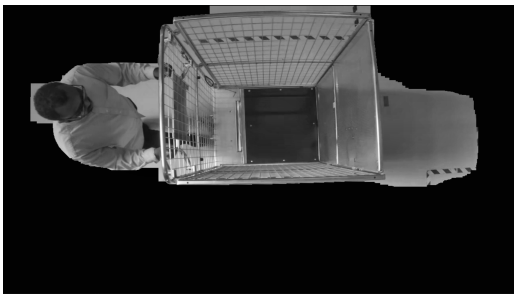
```
1  backSub = cv2.createBackgroundSubtractorKNN(detectShadows=False)
2  kernel = np.ones((5,5), np.uint8)
3  Foreground_Mask = backSub.apply(cframe)
4  Foreground_Mask = cv2.morphologyEx(Foreground_Mask, cv2.MORPH_CLOSE, kernel,
   ↪  iterations=10)
5  Foreground_Mask = cv2.morphologyEx(Foreground_Mask, cv2.MORPH_OPEN, kernel, iterations=15)
6  Foreground = cv2.bitwise_and(cframe, cframe, mask = Foreground_Mask)
7  cv2.normalize(Foreground, Foreground, 0, 255, cv2.NORM_MINMAX)
8  ForegroundGrey = cv2.cvtColor(Foreground, cv2.COLOR_BGR2GRAY)
9  clahe = cv2.createCLAHE(clipLimit=1, tileGridSize=(9,9))
10 diff = clahe.apply(ForegroundGrey)
```

listing 6.1. Code used for background subtraction

An example of using background subtraction is shown in figure 6.1. It should be noted how in figure 6.1b, it is only the movable objects (person and cage) that is showcased. However, shadows are also showcased.



(a) The original frame used to background subtract



(b) After finding the foreground and applying CLAHE

Figure 6.1. Comparison between original frame and background subtracted frame

6.2 Masking

Masking is a part of what is described as data manipulation in the flowchart. the implementation of masking is shown in list 6.2.

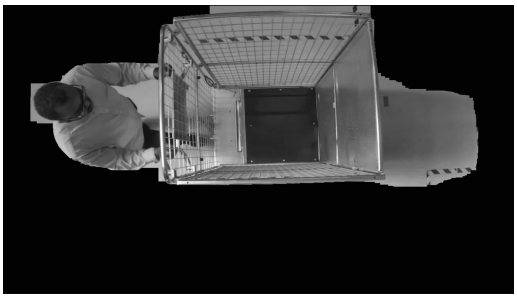
Utilizing the image from background subtraction, the code runs a canny edge detection algorithm to locate edges. The edges are then altered using morphology dilation. The image from background subtraction is then separately run through an array of filtering operations, including bilateral, median and Gaussian blur, and converted to HSV color space. The image is then masked, sorting for gray elements, and converted to a binarized image. this image is then run through morphology operators to remove noise. The image is then run through a `BitwiseAND` with the edge detection from earlier. Another round of morphology is applied to smooth the image. This still binarized image is then used on the CLAHE image from earlier to identify the final Region Of Interest. This is then thresholded and run through another round of morphology.

```
1  kernel = np.ones((5,5), np.uint8)
2
3  edge = cv2.Canny(diff, 150, 160, apertureSize=3, L2gradient = True)
4  edge = cv2.morphologyEx(edge, cv2.MORPH_DILATE, kernel, iterations=4)
5
6  Foreground = filtering(Foreground)
7  hsv = cv2.cvtColor(Foreground, cv2.COLOR_BGR2HSV)
8  hsv = filtering(hsv)
9
10 Lower_Grey = np.array([0, 10, 30])
11 Upper_Grey = np.array([180, 40, 200])
12
13 GreyMask = cv2.inRange(hsv, Lower_Grey, Upper_Grey)
14 GreyMask = cv2.morphologyEx(GreyMask, cv2.MORPH_CLOSE, kernel, iterations=4)
15 GreyMask = cv2.morphologyEx(GreyMask, cv2.MORPH_OPEN, kernel, iterations=4)
16
17 Combined_Mask = cv2.bitwise_and(GreyMask, edge)
18 Combined_Mask = cv2.morphologyEx(Combined_Mask, cv2.MORPH_CLOSE, kernel, iterations=4)
19 Combined_Mask = cv2.morphologyEx(Combined_Mask, cv2.MORPH_ERODE, kernel, iterations=4)
20
21 frame = filtering(diff)
```

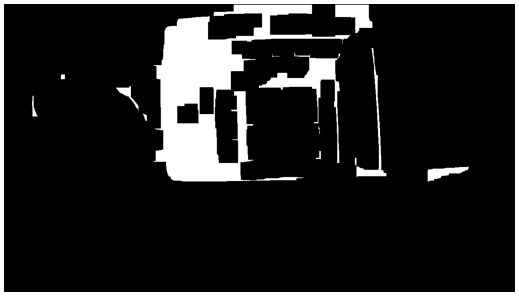
```
22 Cage_Isolated = cv2.bitwise_and(frame, frame, mask = Combined_Mask)
23
24 thresha = cv2.threshold(Cage_Isolated, 140, 255, cv2.THRESH_TOZERO_INV)[1]
25 thresh = cv2.threshold(thresha, 20, 255, cv2.THRESH_BINARY)[1]
26
27 closening = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=60)
28 opening = cv2.morphologyEx(closening, cv2.MORPH_OPEN, kernel, iterations=67)
```

listing 6.2. Code used for masking the image

An example of the code is shown in figure 6.2. Using the input image from figure 6.2a, the masking process is used which as shown in figure 6.2b removes most parts of the person. However, the cage itself misses several parts prior to it being complete. Therefore, by using the second round of morphology as described and shown in figure 6.2c, the person is removed and the region of cage is fully completed.



(a) Frame after applying CLAHE on background subtraction



(b) The combined mask after finding the edge and gray mask

(c) The combined mask after cage isolation thresholding and morphology

Figure 6.2. The process of cumulative masking to isolate the cage

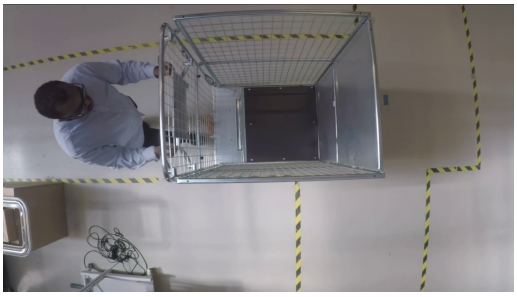
6.2.1 Contours

This segment of code described in list 6.3 is utilized to find contours in the region of interest. It utilized the final output of the last segment, and runs it through OpenCV2 `findContours`. If any contours are found, the algorithm tries to find the biggest area outlined by contours, and finds a bounding rectangle. of this rectangle is the right size to be a cage, it is then drawn on the image. Finally the area inside the rectangle is converted to HSV, and filtered using the filtering process described earlier in the chapter. Additionally a debug mask is generated for troubleshooting purposes.

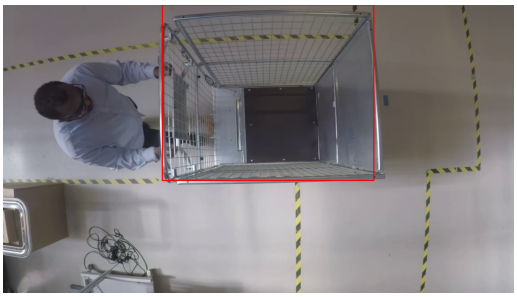
```
kernel = np.ones((5,5), np.uint8)
Contours = cv2.findContours(opening, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

if Contours[1] is not None:

    Contours = Contours[0] if len(Contours) == 2 else Contours[1]
    c = max(Contours, key = cv2.contourArea)
    x,y,w,h = cv2.boundingRect(c)

    if (w < 1100) and (h < 600) and (w >= 300) and (h > 200):
        cv2.rectangle(bgr, (x, y), (x + w, y + h), (0,0,255), 2)
        hsv_Box = cv2.cvtColor(cframe, cv2.COLOR_BGR2HSV)
        hsv_Box = filtering(hsv_Box)
        Box_ROI = hsv_Box[y:y + h, x:x + w]
        res, debug = findbox(Box_ROI, kernel, bgr, x,y,w,h)
```

listing 6.3. Code used for contours

Using the same example as figure 6.1 and 6.2, an example showcases how the contours are then drawn onto the original image. The contours are based on the image shown in figure 6.2c and using the contour process, it is then drawn onto the original image as shown in figure 6.3b.



(a) The original frame used to background subtract



(b) Drawing of contour to the original image

Figure 6.3. Example of contours being drawn into the original image

6.2.2 Finding packages for directionality

The task of the code described in list 6.4 is firstly to find a box inside the region of interest. This is done by masking the region of interest for the colors corresponding to the average brown box, which is then dilated to reduce gaps. which is then used to mask over the original frame, in BGR format. This is then converted to grayscale. this is then converted to a binary image, and morphology is used to reduce noise, gaps and sharpen edges. Based on this, the contours are found.

```
findbox(hsv_Box, kernel, bgr, x,y,w,h):
    Lower_box = np.array([5, 80, 60])
    Upper_box = np.array([20, 120, 255])
    mask_box = cv2.inRange(hsv_Box, Lower_box, Upper_box)
```

```
    mask_box = cv2.morphologyEx(mask_box, cv2.MORPH_DILATE, kernel, iterations=4)

    result_box = cv2.bitwise_and(bgr[y:y + h, x:x + w], bgr[y:y + h, x:x + w], mask =
    ↪  mask_box)
    result_box = cv2.cvtColor(result_box, cv2.COLOR_BGR2GRAY)

    thresh_box = cv2.threshold(result_box, 1, 255, cv2.THRESH_BINARY)[1]
    closening = cv2.morphologyEx(thresh_box, cv2.MORPH_CLOSE, kernel, iterations=6)
    opening = cv2.morphologyEx(closening, cv2.MORPH_OPEN, kernel, iterations=8)
    closening = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel, iterations=8)

    Box_Contours = cv2.findContours(closening, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

listing 6.4. Code used for finding packages inside ROI

If any contours were found, the code described in list 6.5, try to identify the biggest area bounded by them. Then it calculates a bounding rectangle and draws it. This is used to calculate the center of the box and draw a circle around the center. If the center is too close to the edge, it is cut off.

```
cX = 0
    if Box_Contours[1] is not None:
        print("Cage with box")
        Box_Contours = Box_Contours[0] if len(Box_Contours) == 2 else Box_Contours[1]
        bc = max(Box_Contours, key = cv2.contourArea)
        bx,by,bw,bh = cv2.boundingRect(bc)
        if (bw >= 50) and (bh > 50):
            cv2.rectangle(bgr[y:y + h, x:x + w], (bx, by), (bx + bw, by + bh), (0,0,255),
            ↪  2)
            M = cv2.moments(bc)
            cX = int(M["m10"] / M["m00"])
            cY = int(M["m01"] / M["m00"])
            cv2.circle(bgr[y:y + h, x:x + w], (cX, cY), 7, (255, 255, 255), -1)

    if cX == 0 or 99 > (x + cX) or (x + cX) > 1179: return None, None

    return (x + cX), None
```

listing 6.5. Code used for finding center of a package

The third part of the code is shown in list 6.6: If an X coordinate for a box is registered, it is appended to `pts` and the watchdog timer is reset. If `pts` contains any x coordinates the watchdog timer increments. if it exceeds 10, the cage is assumed to have passed, and we start utilizing the data gathered. If `pts` is shorter than 12 entries, we write an error to the console, to inform the user of the critically small sample size, as well as changing the text color to red. If it is less than 22 we warn the user of the small sample size, and color the terminal yellow. if it is 22 or above, we color it green. we then sort the data in `pts`, by drawing vectors between points offset by half of `pts`, to reduce the influence of noise. these vectors are then sorted based on whether they point left or right as votes

for either direction. if the sample is inconclusive we inform the user that the data may be unreliable, and color the terminal yellow, if it is not already red. We then print the results from the vote, as well as the estimated directionality, and reset the terminal color afterwards. Finally all relevant variables are reset, so they are ready for another object.

```
res, diX, debug = pending_task.popleft().get()
if diX is not None:
    pts.append(diX)
    watchdog = 0

    if len(pts) > 0:
        watchdog += 1

        if watchdog > 10:
            Warningtag = 0
            if len(pts) <= 11:
                print('\033[91m'+"ERROR: Sample size is too small") #Color red
                        Warningtag = 1
            elif len(pts) <= 21:
                print('\033[93m'+"WARNING: Sample size is small") #Color orange
            else:
                print('\033[92m') #Color green

            for i in range((len(pts)//2)):
                dX = pts[i+len(pts)//2] - pts[i]
                if dX >= 30:
                    right += 1
                elif dX <= -30:
                    left += 1
            if (0.5 <= right / (left + 1) <= 2) or (0.5 <= left / (right + 1) <= 2):
                if  Warningtag != 1:
                    print('\033[93m') #Color orange
                print("WARNING: Result unreliable")
            print(left,"|",right)
            dirX = "Moving left" if left > right else "Moving right" if left < right else
            ↪  "Movement not determined"
            print(dirX+'\033[0m') #Color reset
            dX = 0
            right = 0
            left = 0
            pts = []
```

listing 6.6. Code used to describe directionality of a package

Tests 7

Based on the implementation described in chapter 6, it is important to evaluate if the detection and directionality algorithm is satisfactory to the requirement specification designated in chapter 4.

7.1 Test environment and setup

The tests for the evaluation of the system is done in Lyngsoe Systems' warehouse, which has plenty of light that is not constant because natural light influences the scene. Therefore the test conditions are not ideal, but this could also be seen as an advantage since if the system can work correctly most of the time in bad condition, it should also do it in better circumstances.

Regarding the test setup, the following materials will be used to the testing process:

- 1 camera
- 2 cages
- 1 pallet
- Packages

The camera used to the tests is the GoPro Hero 5 described in implementation in chapter 6. The two cages are used to run through the camera area as shown in figure 7.1 and will be described as **Cage 1** and **Cage 2** as shown in figure 7.1a and 7.1b respectively. During the tests, the cages will be loaded or emptied with packages.

(a) Cage 1

(b) Cage 2

Figure 7.1. Cages used to tests

The directionality algorithm is set to sort out any results below 10 votes. The reason, 10 votes were chosen, is due to the fact that the videos are running in 15 fps, and as such, a

total of 15 votes can be generated every 2 seconds. This means that a cage needs to be detected for at least 1,33 seconds to be counted. During development, we had noted that most good measurements were in the ballpark of 15-35 votes, whereas noise and things that should not be detected rarely exceeded 5 votes. as such it has been placed in between the 5 and 15.

If the data is inconclusive, such as having equally many votes for either direction, or close to it, these are flagged separately, but are not counted as valid in the test.

The outcome of the prototype is shown as the code terminal and displayed. Here, the direction votes are counted and described, and then the outcome is showcased as well. Finally, to test the prototype if it complies with the requirements set, 2 evaluation methods are presented and then tested on in this chapter.

To do these evaluations truth tables will be used. Based on our system being split in two parts, namely the identification of the cage, and the registering of movement, they will be described separately, however movement is dependent on detection, so it will be described using logical operators, as opposed to a truth table.

First we will describe the classification of Detection. True positives are classified as the identification of a cage with packages. A false positive is classified as the identification of an object that is not a cage with packages. True negatives are classified as not identifying a cage with packages when one is not present. A false negative is classified as not identifying a cage with packages when one is present. This can be illustrated in a truth table, as seen in 7.1

Detection	Output	Classification
T	T	True Positive
T	F	False Negative
F	T	False Positive
F	F	True Negative

Table 7.1. Truth table illustrating all possible outcomes for the detection of a cage with packages

Using these classification it is then possible to find a set of TPR (True Positive Rate) and FPR (False Positive Rate) with equations 7.1

$$\text{TPR} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}} \tag{7.1a}$$

$$\text{FPR} = \frac{\text{False positives}}{\text{False positives} + \text{True negatives}} \tag{7.1b}$$

The movement part has the following classification: True positives are classified as determining the direction of a cage with packages correctly. A false positive is classified as determining the direction of a cage with packages incorrectly. True negatives are classified as not seeing any movement when no cage with packages is moving. A false negative is classified as not seeing any movement when a cage with packages is moving. As this is dependent on the entity being a cage with packages, it can be described with the logical statement seen in 7.2

$$\neg(A \longrightarrow \neg B) \tag{7.2a}$$

where A is the detection of a cage with packages, and B is the movement of said cage. A table illustrating the possible outcomes of this has been constructed as seen in 7.2

Detection	Movement	Output	Classification
T	T	T	True Positive
T	T	F	False Negative
T	F	T	False Positive
T	F	F	True Negative
F	T	T	False Positive
F	T	F	True Negative
F	F	T	False Positive
F	F	F	True Negative

Table 7.2. Truth table illustrating all possible outcomes for 7.2

This table in 7.2 will be used as the de facto reference for evaluating the tests

7.2 Subtest

The first evaluation method that will be tested on is subtesting. Here there will be tested on small video sequences where the result is known and therefore can be tested against the classification of the results i.e., how many true positives, false positive, true negatives, and false negatives are outputted, as seen in 7.2

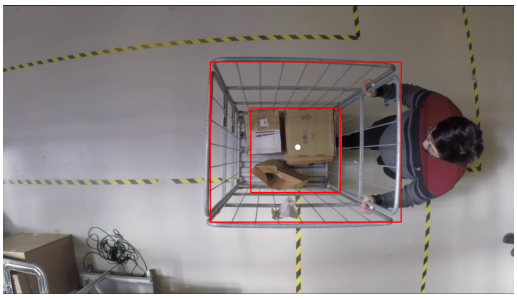
The video sequences that will be tested against are a series of 12 video clips with varying length from 4 seconds to 16 seconds. These video clips contain:

- 2 different cages which includes 1 video sequence moving left and 1 moving right equaling 4 video sequences.
- 2 video sequences of a single cage with few packages moving both directions.
- 2 video sequences with an empty cage moving both directions.
- 2 video sequences of a person moving both direction.
- 2 video sequences of a pallet moving both directions.

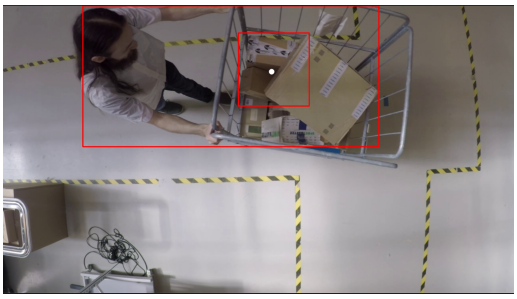
All video sequences can be seen on Github under the VIDEO_CLIPS folder

7.2.1 Cage 1 with packages

The first set of test is 2 video sequences of a cage that is filled with packages and another 2 video sequences of the same cage with few packages. The expected output from the video sequences is getting at least 10 votes for the direction the cage is moving, where there will be 1 moving left and 1 moving right for when the cage is filled with packages and when there is a few packages. The resultant output image can be seen in figure 7.2



(a) Output image from the detection algorithm for cage 1 with few packages



(b) Output image from the detection algorithm for cage 1 filled with packages

Figure 7.2. Output image for cage 1

At sub figure 7.2a can be seen a cage with a few packages moving left and in sub figure 7.2b can be seen a cage filled with packages moving right. The big red box that can be seen in both figures is the approximate location of the cage detected with our algorithm. The smaller red box is the approximate location of the package found with the white dot being the middle of the red box used for finding the direction of the cage. The output from these 4 video sequences can be seen in listing 7.1.

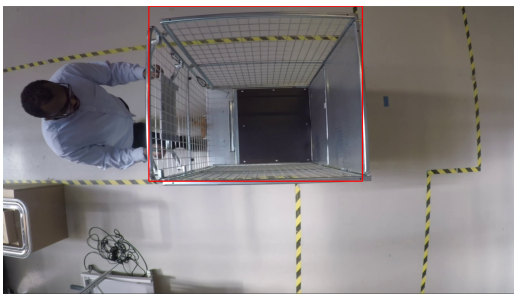
```
cage1-fewpackages left
27 | 0
Moving left

cage1-fewpackages right
0 | 12
Moving right

cage1-packages left
25 | 0
Moving left

cage1-packages right
0 | 23
Moving right
```

listing 7.1. Output from the terminal for the 4 video sequences

It can be concluded from the output that the algorithm can find the direction of the cage and that all votes for the direction is to the correct one making the findings true positives. Although in test, `cage1-fewpackages left` the tallied vote is 12 which is substantially lower than the other 3 video sequences. This can be possibly attributed to the speed at which the cage was moving or a few frames the algorithm could not find anything. Looking through the video sequence, it is more likely that the cage was moving too fast. Therefore there were just a few frames where it was possible to find the cage which reduced the votes for the direction. This can possibly be mitigated with a higher frame rate camera.

7.2.2 Cage 2

Another 4 set of test has been done on cage 2, where 2 sequences are of an empty cage and the other 2 are the cage moving right or left. The expected output from the empty cage video sequences is getting less than 10 votes or nothing for the direction the cage is moving. The expected output for the cage with packages is getting at least 10 votes for the direction it is moving where there will be 1 moving left and 1 moving right. The resultant output image can be seen in figure 7.3



(a) Output image from the detection algorithm for empty cage 2

(b) Output image from the detection algorithm for cage 2 filled with packages

Figure 7.3. Output images for cage 2

At sub figure 7.3a can be seen a cage moving right and in sub figure 7.3b can be seen a cage filled with packages moving left. The output from the 4 video sequences can be seen in listing 7.2

```
cage2-empty right
Nothing

cage2-empty right 2
Nothing

cage2-packages left
11 | 0
Moving left

cage2-packages right
0 | 10
Moving right
```

listing 7.2. Output from the terminal for the 4 video sequences

It can be concluded from the output that the algorithm does not find a direction for an empty cage meaning it has given a true negative. It is also seen that it can find the direction of the cage with packages and that all votes for the direction is to the correct one making the findings true positives. The votes for the direction are again on the low side, and seeing through the video sequences, it is highly correlated with the fact that the cage was moving fast. In a one time occurrence in our testing there was a small error where testing the output for `cage2-packages right` was:

```
WARNING: Sample size is small
0 | 9
Moving right
```

listing 7.3. Output from the terminal for the 4 video sequences

Which would be a False negative. There is a high likelihood that this was a one time occurrence.

7.2.3 Pallet

While the project has been focused on being able to track and determine the directionality of a cage with packages, it is important to verify that the cage detection system does not detect and determine the directionality of other objects. In this case, a pallet has been used, which is a common object in sorting centers as described in section 2.5.

The video set of the pallet is 2 video sequences, where in the first sequence the pallet is going left and in the second sequence, it is going right. The expected output is based on the video sequences the system will not detect the pallet and thereby no results are given.

```
pallet left
ERROR: Sample size is too small
0 | 0
Movement not determined

ERROR: Sample size is too small
WARNING: Result unreliable
0 | 1
Moving right

pallet right
nothing
```

listing 7.4. Output from the terminal for the video sequences of a pallet

(a) Screenshot of output image of pallet going left

(b) Screenshot of output image of pallet going right

Figure 7.4. Output image from the detection algorithm for a pallet

In the first video sequence, it can be seen in the output from the terminal shown in listing 7.4 and sub figure 7.4a that the system detected the pallet. This is most likely due to background subtraction, shadows generating noise, and the pallet itself having the same colors as packages. Although since the tallied votes are less than 10 this makes the output a true negative. The output of second video sequence performs as desired as shown in list 7.4 and is wholly a true negative.

7.2.4 Person

It is also decided to test if one person walks through the test environment. This is to test if the system detects and track the directionality of personnel. The test is performed by using two video sequences, where the first sequence includes a person walking from left to right with a package, while the second sequence is a person walking from right to left without any packages. The expected outcome of this test is that the video sequences get less than 10 votes or ideally nothing. Therefore, it is expected that the system does not detect and therefore determine the directionality of the person.

```
person right
Nothing

person left
Nothing
```

listing 7.5. Output from the terminal for the video sequences of a person

(a) Screenshot of output image of person going right

(b) Screenshot of output image of person going left

Figure 7.5. Output image from the detection algorithm for a person

It can be seen in list 7.5 and figure 7.5 that the system does not detect anything as desired. Thereby, it can be confirmed that the system do not track and determine the directionality of a person and that the output is a true negative.

7.2.5 Conclusion of subtest

Testing trough the 12 video sequences the results that were given is 6 true positives and 6 true negatives as seen in table 7.2.5.

Test number	Video sequence	Expected result	Result	Classification
1	Cage 1 few packages	Left	Moving left	True Positive
2	Cage 1 few packages	Right	Moving right	True Positive
3	Cage 1 full	Left	Moving left	True Positive
4	Cage 1 full	Right	Moving right	True Positive
5	Empty Cage 2	Nothing	Nothing	True Negative
6	Empty Cage 2	Nothing	Nothing	True Negative
7	Cage 2 full	Left	Moving Left	True Positive
8	Cage 2 full	Right	Moving right	True Positive
9	Pallet	Nothing	1 vote right	True Negative
10	Pallet	Nothing	Nothing	True Negative
11	Personnel	Nothing	Nothing	True Negative
12	Personnel	Nothing	Nothing	True Negative

Table 7.3. Table describing the different objects, the expected result, the result given from the system, and the classification

This makes the true positive rate 1 and false positive rate 0 as seen in equation 7.3

$$\text{TPR} = \frac{6}{6+0} = 1 \tag{7.3a}$$

$$\text{FPR} = \frac{0}{6} = 0 \tag{7.3b}$$

As seen in the conclusion in the test for cage 1 and cage 2 there were cases where the votes for the direction were on the low side which was concluded to be because the speed of the cage was too fast for 15 frames per second. This can be mitigated by having a higher frame rate camera, which would make more points to detect the direction. Although this would also mean that there would be more calculation made, that will take longer to find the direction. In general the speed of the cages in these video sequences is faster than what happens in reality in a sorting center, because there are safety requirements limiting the speed of moving a cage.

7.3 System test

Based on the subtests made and described in section 7.2, a system test is accomplished to determine if the system complies with the requirement specification described in chapter 4. The system test contains 1 long video sequence that has a length of 5 minutes and 39 seconds. The video sequence is named "15FPS_720PL.mp4" and can be seen on Github. From this a table can be made for the system test as seen in table 7.4.

Object order	Object type	Expected result	Result	Classification
1	Cage 1	Left	Moving left	True Positive
2	Cage 1	Right	Moving right	True Positive
3	Empty Cage 2	Nothing	Nothing	True Negative
4	Empty Cage 2	Nothing	Nothing	True Negative
5	Pallet	Nothing	Moving left	False Positive
6	Personnel	Nothing	Nothing	True Negative
7	Empty Cage 2	Nothing	Nothing	True Negative
8	Pallet	Nothing	Moving left	False Positive
9	Cage 1	Left	Moving left	True Positive
10	Noise	Nothing	ERROR: Sample size too small	True Negative
11	Cage 1	Right	Moving right	True Positive
12	Empty Cage 2	Nothing	Nothing	True Negative
13	Rolling desk	Nothing	ERROR: Sample size too small	True Negative
14	Cage 1 up	Nothing	Moving Right	False Positive
15	Cage 1	Right	Moving Right	True Positive
16	Personnel	Nothing	Nothing	True Negative
17	Personnel	Nothing	Nothing	True Negative
18	Cage 1 up	Nothing	Moving left	False Positive
19	Cage and pallet	Left	Moving left	True Positive
20	Cage and pallet	Left	Moving left	True Positive
21	Noise	Nothing	ERROR: Sample size too small	True Negative
22	Pallet	Nothing	Nothing	True Negative
23	Cage 1+2	Left	Moving Left	True Positive
24	Empty Cage 2	Nothing	ERROR: Sample size too small	True Negative
25	Cage 2	Left	WARNING: Sample size small, Moving left	False Negative
26	Cage 1	Left	Moving left	True Positive
27	Cage 1+2	Right	Moving right	True Positive
28	Cage 2	Right	WARNING: Sample size small Moving right	False Negative
29	Cage 2 stop and move	Left	Moving left	True Positive
30	Personnel	Nothing	Nothing	True Negative
31	Cage 2	Right	Moving right	True Positive

Table 7.4. Table describing the different objects, the expected result, the result given from the system, and the classification of the system test

Based on table 7.4, it can be determined that the system can struggle to differentiate between cages with packages, and pallets. From the table it is also possible to determine how many True positives, False Positives, True Negatives, and False Negatives there are. Using those number it is possible to calculate the TPR and FPR seen in equation 7.4

$$\text{TPR} = \frac{12}{12+2} = 0.85 \tag{7.4a}$$

$$\text{FPR} = \frac{4}{4+13} = 0.23 \tag{7.4b}$$

From equation 7.4 it can be concluded that the system can determine the direction of the entity detected with a high level of accuracy for a prototype. Although the detection code produces false positives more than desired, but is able to reliably differentiate personnel from larger elements, and cages with packages from those without. The directionality determination code operates with a low error rate, and the errors are mostly flagged as unreliable by the system, making them easy to distinguish. When compared to the requirement specification, the results are shown in the following table 7.5.

Requirement	**Status**
The camera must always be running	FALSE
The image resolution must be set to 1280x720 pixels	TRUE
The camera must have a maximum frame rate at 15 fps	TRUE
The computer must be able to receive data from the camera	TRUE
The computer must be able to process the camera data	TRUE
The computer must be able to object-detect the cage	TRUE
The computer must detect and store the direction of the cage	TRUE

Table 7.5. Checklist of system requirements and whether they are achieved.

The system is not able to run on a live feed from the camera, and as such fails this condition. It should be noted that the system is theoretically able to do so by changing only one parameter, however it is not configured as such. While the system is able to receive data from the camera, it does not do this automatically. As such, this one is only conditionally true.
The computer is able to to detect the directionality of the cage, however as of the current implementation, this is only output to the terminal, and technically not stored. This is therefore partially true.

8 Discussion

This chapter will discuss the choices made in the development of the system, how this compares to the ideal, and the shortcomings of our implementation.

With regards to the problem statement in section 2.7, the developed prototype is able to track and determine the directionality of a cage with packages. While the prototype can find most cages and packages, the system is inconsistent in detecting the cage and therefore sometimes finds the directionality of other objects with similar color characteristics as a cage such as a pallet. A reason for this inconsistent behavior can be attributed to background subtraction. As described in chapter 7, the light conditions in the test environment were inconsistent. Therefore, it is entirely possible that the background subtraction process could give unpredictable results attributing to the inconsistency. Regarding background subtraction, the system test video did not have any downtime and also because we did not implement the history feature to make the background model, it did not work as intended. Therefore, if there were downtime in the video, such that the background model could be made, the prototype would be more consistent to detect cages.

In chapter 7, two different evaluation methods have been used: Sub tests and system test. It is shown in section 7.2.5, that the results of sub tests have been highly successful compared to the system tests. This difference can be attributed to sub tests being isolated cases and as such the results are much more polarized, and they are also subject to selection bias. In system tests, as described above, there is several potential points of failure and as such the odds of a flawless test is much less likely.

In terms of our truth table 7.2 our system considers three different type of outputs as false positives. While this is logically true, the three are different from one another, and can be utilized to identify the weakest link in the system. For example entry "True-False-True", which is the ordinary false positive shows inconsistency in the ability to detect movement. In comparison entry "False-True-True" shows complications in identifying whether a cage contains packages, or is even a cage, in other words the object detection algorithm. Entry "False-False-True" shows a complete failure of the system, which is worst case. This could have been utilized to troubleshoot, or measure the performance of the system more accurately.

For the sake of simplicity the cutoff regions mentioned in 5.2 were never implemented. During testing we found that the watchdog timer was very consistent on its own, and as such has been utilized to handle the termination process alone. This does create a few edge cases, which could potentially give wrong outputs, however this has not been a big issue in our testing.

During testing a minor, but not insignificant bug was found. The final computations calculate based on vectors generated from a list of points, and all vectors below a certain length are not considered. The code for determining the correctness of a set of measurements is based on only the points however, and as such the code may occasionally determine a measurement to be valid, despite having less votes than the determined threshold for being reliable. Fixing this is not hard, but doing so would invalidate all tests: Due to time limitation, this was left out. The code can be seen in list 6.6.

Overall, the prototype has addressed the issue outlined by the problem statement, and is compliant with most of the requirements. The requirements, the prototype is not compliant with, can easily be implemented as described in section 7.3.

At the start of development we had no knowledge of the existence of optical flow. We later learned of this function, which could have streamlined a large part of the development. At the time however, integrating optical flow would not provide additional functionality to the system, since a means of determining directionality was already developed, and as such it was not implemented. In retrospect utilizing optical flow could provide a higher level of accuracy, at a mentionable performance hit, and as such is a more ideal approach, given sufficient processing power. It is also possible to use it as supplementary for measurements that are inconsistent.

Conclusion 9

The project started presenting the numbers of package shipping globally and nationally: In Denmark in 2020, the total amount of packages sent to or inside the country were 170 million packages. There were 11.810 complaints from customers to the shipping company, where 54% of those received compensation as result. The compensation in total was stated to be 63 million DKK. Due to the fact that compensation can be different (i.e., resent packages, financial etc.), Lyngsoe Systems estimates that the financial compensation is approximately 25 million DKK a year. Therefore, Lyngsoe Systems has proposed the problem and an initial problem statement has been formed:

How can vision technology be used to detect an object and how can the directionality of the object be found?

Using the initial problem statement, the environment of sorting centers and cages has been researched and based on that, relevant image processing techniques have been discussed. Therefore, it is concluded it is feasible to detect and find the direction of cages. Based on this, the problem statement has been formulated:

How can computer vision be implemented to track and determine the directionality of a cage with packages?

Based on our problem statement, research has been conducted on state of the art techniques in computer vision. The research has shown that it is possible to use edge detection and color masking to find cages. Blur techniques and histogram stretching and equalization is found to make the system more reliable. From this, the prototype system can be designed into two parts: Detection of the cage and determining the directionality of the cage.

After implementing and testing the prototype system, it is concluded that the prototype fulfills majority of the system requirements which has been outlined, where those that have not been completed, can easily be implemented. The system has a high true positive rate of 85%, meaning that it can find cages with packages 85% of the time. However, the false positive rate, meaning the amount of times the direction of an incorrect object has been found, is also larger than what would be ideal. Showing that the system is more likely to return a positive result rather than a negative, regardless of whether it is true. The low false negative rate also supports this argument. This means that the prototype is unreliable in detecting the correct object, but the directionality algorithm is consistent. It can then be concluded that the prototype mostly fulfills the problem statement.

Bibliography

Adobe, **2022**. Adobe. *Understanding shutter speed, both slow and fast.*, 2022. URL https://www.adobe.com/creativecloud/photography/discover/shutter-speed.html.

Boateng et al., **01 2012**. Kwame Boateng, Benjamin Weyori and David Laar. *Improving the Effectiveness of the Median Filter*. International Journal of Electronics and Communication Engineering, 5, 85–97, 2012.

Boesch, **2022**. Gaudenz Boesch. *What is OpenCV? The Guide for Computer Vision Development in 2022*, 2022. URL https://viso.ai/computer-vision/opencv/.

Danish Standards, **2021**. Danish Standards. *DS/EN 12464-1:2021 (Light and lighting - Lighting of workplaces - Indoor work places)*, 2021.

DCARA, **2020**. DCARA. *Redegørelse for 2020 om det danske postmarkeds tilstand og udvikling*, 2020. URL https://www.trafikstyrelsen.dk/da/-/media/TBST-DA/Post/Lister/Publikationer/Redeg%C3%B8relse-for-2020-om-det-danske-postmarkeds-tilstand-og-udvikling.pdf.

Du, **05 2020**. Shuchen Du. *Understanding Deep Self-attention Mechanism in Convolution Neural Networks*, 2020. URL https://medium.com/ai-salon/understanding-deep-self-attention-mechanism-in-convolution-neural-networks-e8f9c01cb251.

EMAG Technologies, **2017**. EMAG Technologies. *EM_wave.png (734×290)*, 2017. URL http://www.emagtech.com/wiki/images/e/e7/EM_wave.png.

Lumen, **2018**. Lumen. *Electromagnetic Waves and their Properties | Boundless Physics*, 2018. URL https://courses.lumenlearning.com/boundless-physics/chapter/electromagnetic-waves-and-their-properties/.

Mamdouch, **2020**. Tarek Mamdouch. *Color spaces (RGB vs HSV) - Which one you should use?*, 2020. URL https://discover.hubpages.com/technology/Color-spaces-RGB-vs-HSV-Which-one-to-use.

Mansurov, **2019**. Nasim Mansurov. *What is ISO? The Complete Guide for Beginners*, 2019. URL https://photographylife.com/what-is-iso-in-photography.

Mansurov, **2021**. Nasim Mansurov. *Introduction to Shutter Speed in Photography*, 2021. URL https://photographylife.com/what-is-shutter-speed-in-photography.

Moeslund, **2012**. Thomas B Moeslund. *Introduction to Video and Image Processing: Building Real Systems and Applications*. Undergraduate topics in computer science. Springer London, Limited, London, 2012. ISBN 1447125029.

Nikon, **2011**. Nikon. *ISO Control*, 2011. URL https://www.nikonusa.com/en/learn-and-explore/a/products-and-innovation/iso-control.html.

OpenCV, **2021**. OpenCV. *OpenCV: Contours : Getting Started*, 2021. URL https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html.

Paris et al., **2008**. Sylvian Paris, Pierre Kornprobst, Jack Tumblin and Frédo Durand. *A Gentle Introduction to Bilateral Filtering and its Applications*, 2008. URL https://people.csail.mit.edu/sparis/bf_course/course_notes.pdf.

Post Danmark A/S, **2020**. Post Danmark A/S. *Årsrapport 2020*, 2020. URL https://www.postnord.dk/siteassets/pdf/finansiell-information/arsregnskabsmeddelser/aarsrapport_2020.pdf.

SCM, **2014**. SCM. *PostNord samler pakkerne i Danmark | SCM.dk*, 2014. URL https://scm.dk/postnord-samler-pakkerne-i-danmark.

Srivastava, **2020**. Smriti Srivastava. *OpenCV Vs MATLAB: Which Is Best For Successful Computer Vision Project?*, 2020. URL https://www.analyticsinsight.net/opencv-vs-matlab-which-is-best-for-successful-computer-vision-project/.

Stanford University, **2020a**. Stanford University. *Lossless Data Compression*, 2020a. URL https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossless/index.htm.

Stanford University, **2020b**. Stanford University. *Lossy Data Compression*, 2020b. URL https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossy/index.htm.

Statista, **2021**. Statista. *Chart: The Parcel Shipping Boom Continues*, 2021. URL https://www.statista.com/chart/10922/parcel-shipping-volume-and-parcel-spend-in-selected-countries/.

Statista, **2022**. Statista. *E-commerce worldwide - statistics & facts*, 2022. URL https://www.statista.com/topics/871/online-shopping/.

Statistics Denmark, **2020**. Statistics Denmark. *Rekordmange handler på nettet*, 2020. URL https://www.dst.dk/da/presse/Pressemeddelelser/2020/2020-03-06-rekordmange-handler-paa-nettet.

Sudhakar, **2017**. Shreenidhi Sudhakar. *Histogram Equalization | Towards Data Science*, 2017. URL https://towardsdatascience.com/histogram-equalization-5d1013626e64.

US Global Mail, **2022**. US Global Mail. *Parcel Sorting Center - US Global Mail*, 2022. URL https://www.usglobalmail.com/parcel-sorting-center/.

Verhoeven, **2017**. Geert Verhoeven. *The reflection of two fields – Electromagnetic radiation and its role in (aerial) imaging*. AARGnews, 55, 13–18, 2017. doi: 10.5281/zenodo.3534245.

Appendix A

A.1 Electromagnetic waves

To understand how a camera can capture an image, it is important to account for electromagnetic waves. An electromagnetic wave (or EM wave for short) is defined as a wave that contains an electric field (E-field) and a magnetic field (B-field). The energy in the wave is stored in both fields. An EM wave is also known to be transverse, meaning that the two fields are perpendicular to each other while also being perpendicular to the direction which the wave travels as shown in figure A.1. EM waves can therefore also be described as photons [Lumen, 2018].

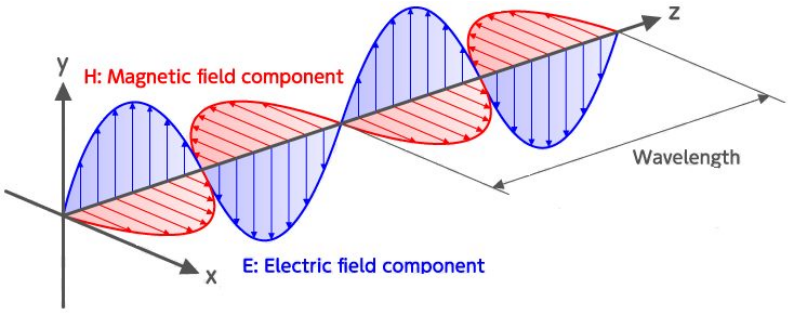


Figure A.1. An electromagnetic wave [EMAG Technologies, 2017]

EM waves can be expressed in terms of energy, wavelength and frequency. However, when comparing or classifying EM waves, the used parameter is generally wavelength which is measured in meters [m]. EM waves are classified into seven different types: Radio waves, microwaves, infrared, visible light, ultraviolet, X-rays and gamma rays. This is also shown in figure A.2.

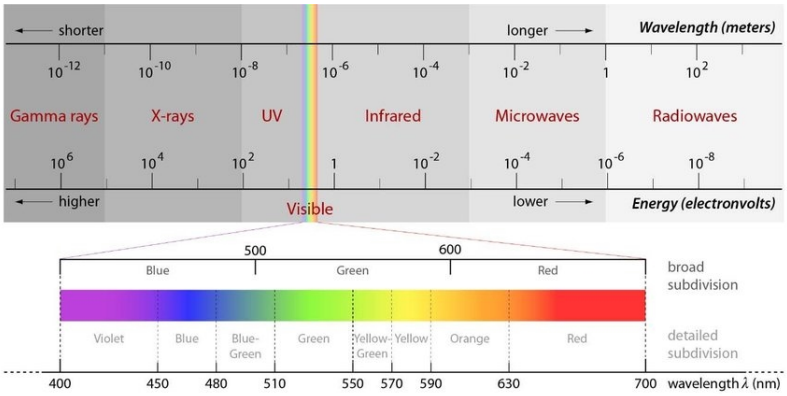


Figure A.2. Electromagnetic spectrum [Verhoeven, 2017]

The different classification has its usability: As an example, As shown in figure A.2, the human eye can only see a limited area of electromagnetic spectrum known as the visible spectrum. The range of the spectrum is approximately from 400 nm to 700 nm. [Verhoeven, 2017]. Since the human eye is limited to seeing light in this spectrum, it is also set as the range that a camera should capture. While cameras that capture outside of the visible spectrum (such as infrared or ultraviolet light) exist, it will mainly be focused on cameras that can capture light inside the visible spectrum.

A.2 Camera

Aperture

The principle behind aperture is to limit the visibility of the lens. By doing so, the angles of which the light can enter the camera is vastly reduced. This increases the depth of field, as the image shifts further from a perspective view, and more towards an orthographic view. It is in practice not possible for the camera to transition entirely, but the closer it gets, the further the depth of field expands, along with the focal length narrowing. The major trade-off for this is the sharp reduction in the light entering the camera. This can be compensated for with a longer shutter speed.

Shutter speed

The shutter speed of a camera is determined by how long the shutters stay open for the camera. By adjusting the shutter speed, you can manipulate how much light enters the camera, and hits the sensors, thereby altering the brightness of the image. This can be useful when working in a dark environment, or working with very bright elements, such as the sun. The downside of working with extended exposure times is motion blur [Mansurov, 2021]. Whenever something is moving, The extended exposure time will make it leave a trail in it's path. This can be desired for artistic reasons, however for image processing, having this blurred trail can be a critical flaw for the application, and as a consequence ISO may be a preferred solution to adjust brightness [Adobe, 2022].

ISO

The most common action of ISO (International Organisation of Standardisation) is to brighten or darken pictures, when adjusting the exposure time isn't a viable option. This could be in case of photographing moving elements, or recording video. Adjusting the ISO can alter the image, by Extrapolating imperfections in the captured image, by showing a grainy noise when it is configured high, and by washing out a certain level of fine material texture when configured low [Nikon, 2011]. ISO manipulates the image by altering the data as the sensor inputs are mapped to the image. By working with the raw input, it can produce a much more precise and clear brightening of an image, as compared to what can often be achieved in post processing. ISO Got its name when it's predecessors "ASA" and "DIN" were combined into a single global standard in 1974, adopting the name of the organization that had managed the merging [Mansurov, 2019].

A.2.1 Data properties

File format

Finally, an other important factor in image processing is the size of the image file. A raw digital image contains all data from the image sensor, which means that this type of image files have high quality in regards to details, but will naturally also take up significant memory and storage. This image file is advantageous for editing/manipulating photos, when a very high quality photo is desired. However, for a real-time system, e.g. a surveillance system like a security camera, the bigger file sizes and longer processing times is significantly disadvantageous. For systems, where a smaller file size or faster processing times are needed, it is preferable to compress the image. There exists two types of compression: Lossless and lossy compression.

In lossless compression, the data is compressed such that the original data can be be perfectly reconstructed from the compressed data. Thereby, despite that the file size has been reduced due to the compression, no data has been lost. The compression has been done using several algorithms such as Huffman coding [Stanford University, 2020a]. In comparison to lossless compression, lossy compression focuses on minimizing the file size by removing redundant information. Therefore, some data might get lost in the process, which mean that the original image cannot be reconstructed perfectly based on the lossy compressed data. However, the compression algorithm has been optimized from the human vision's point of view, such that the compressed image looks very similar to the original. Compared to lossless compression, lossy compression is very efficient regarding the file size, where the compressed image's file size can be as small as 10% of the original size without visible degradation of the image quality [Stanford University, 2020b]. In table A.1, the different compression types are compared and their respective advantages and disadvantages have been highlighted. The disadvantages are taken from the point of view of image processing. Here, the preferable outcome is to have a small file size (thereby an efficient compressed image) that do not have any redundant data in the image.

Compression type	**Advantage**	**Disadvantage**	**Example of file type**
No compression (Raw)	Image of highest quality	Big file size Long processing time	.pgm, .ciff
Lossless	High quality image with smaller file size	Unnecessary data	.png, .gif
Lossy	Decent quality image with significantly smaller file size Minimal-to-none redundant data	Heavy compression might degrade the image quailty	.jpeg

Table A.1. The advantage and disadvantage of different compression types