AUTOMATED IMAGE PROCESSING FOR SCRATCH DETECTION ON SPECULAR SURFACES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

VOLKAN OKBAY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2018

Approval of the thesis:

AUTOMATED IMAGE PROCESSING FOR SCRATCH DETECTION ON SPECULAR SURFACES

submitted by **VOLKAN OKBAY** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Tolga Çiloğlu _____
Head of Department, **Electrical and Electronics Engineering**

Prof. Dr. Gözde Bozdağı Akar _____
Supervisor, **Electrical and Electronics Engineering, METU**

Examining Committee Members:

Prof. Dr. İlkay Ulusoy
Electrical and Electronics Engineering, METU _____

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering, METU _____

Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı
Electrical and Electronics Engineering, METU _____

Assist. Prof. Dr. Ulaş Yaman
Mechanical Engineering, METU _____

Assist. Prof. Dr. Osman Serdar Gedik
Computer Engineering, Yıldırım Beyazıt University _____

Date: 07.09.2018

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: Volkan Okbay

Signature :

ABSTRACT

AUTOMATED IMAGE PROCESSING FOR SCRATCH DETECTION ON SPECULAR SURFACES

Okbay, Volkan
M.S., Department of Electrical and Electronics Engineering
Supervisor : Prof. Dr. Gözde Bozdağı Akar

September, 151 pages

In industry, problems due to human error, mechanical flaws and transportation may occur; besides, they need to be detected in fast and efficient ways. In order to eliminate failure of human inspection, automated systems come in action, usually image processing involved. This thesis work, targets one common mass production problem on specular surfaces, i.e. scratch detection. To achieve this, we have implemented two different prototypes. The low-cost system is based on basic line detection, and the mid-end system depends on learning based detection.

Both systems are implemented on embedded platforms and performance comparisons are done. Detailed analysis is carried out on computational cost and detection performance. This real-world episode is done on a mechanical prototype in laboratory environment.

Keywords: Scratch Detection, Specular Metal, Reflective, Industrial Surface Inspection, Machine Learning, Hough Transform, Convolutional Neural Networks, Image Processing, Machine Vision

ÖZ

METALİK YANSIMALI YÜZEYLERDE OTOMATİK ÇİZİK TESPİTİ İÇİN GÖRÜNTÜ İŞLEME SİSTEMİ

Okbay, Volkan
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Gözde Bozdağı Akar

Eylül 2018 , 151 sayfa

Endüstriyel alanda, insan, mekanizma veya ulaştırma kaynaklı oluşabilecek bazı problemlerin hızlı ve verimli şekilde tespit edilmesi önem kazanmıştır. Bu tespit sırasında yine insan faktörünü aradan çıkarmak için genellikle görüntü işleme içeren otomatik sistemler kullanılmaktadır. Bizim tez çalışmamızda ise, yansımalı yüzeylerde görülen bu tip bir seri üretim problemine yoğunlaşıldı. Bu tez çalışması ana hatlarıyla iki adet metod sunmaktadır. Özele inmek gerekirse, basit çizgisel objeleri tespit eden düşük maliyetli bir sistem ve orta seviye kartlarda kullanılabilecek bir makine öğrenmesi temelli algoritma. Bu çalışmada, ayrıca literatürdeki bazı yöntemlerin performanslarına değinilmiştir.

Esas amaç verilen özel problemi düşük maliyet ile çözmektir, bu nedenle yöntemlerde basit düşünceler ve donanımı daha az yoracak gerçeklemeler tercih edilmiştir. Yapılan işler arasında savunulan yaklaşımların bilgisayar destekli modelleri de yer almaktadır. Bu şekilde yapılan karşılaştırmalardan sonra, iş odağı gerçek dünyada farklı donanımlara aktarmaya kaymıştır. Bu fiziksel aşama laboratuvarda oluşturulan bir mekanik prototip yardımı ile tamamlanmıştır.

Anahtar Kelimeler: Çizik Tespit, Yansımalı Metal, Endüstriyel Yüzey Taraması, Makine Öğrenmesi, Hough Dönüşümü, Yapay Sinir Ağları, Görüntü İşleme, Makine Görüsü

Dedicated to my friends, my family and my future...

ACKNOWLEDGMENTS

I should start with thanking my supervisor Prof. Dr. Gözde BOZDAĞI AKAR. It is due to her constant support, faith in me and total patience towards me.

Secondly, owing to his support on prototyping and not leaving us alone on factory visits, I should thank Asst. Prof. Ulaş YAMAN, including his students from Mechanical Engineering Department.

Thirdly, I have to mention "ARÇELİK A.Ş. Bulaşık Makinesi İşletmesi" which is an innovative company eager to catch up with the latest industrial developments. They supported us with the interest of their engineers and workers. Moreover, they have provided hardware requested from our side, financial sponsorship and metal sheet samples that are valuable for forming data set.

I am always thankful for my mother Zeynep, my father Taner, my brother Mert and rest of my family having faith in me and encouraging me with freedom.

I wish we will never break with Alican HASARPA and Bora BAYDAR. Thank you for always being there for each other.

Lastly, I am grateful to TUBITAK for their contribution. The scholarship aid they provided without waiting for a return, supplied an important convenience to my education life.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xv

LIST OF FIGURES xvii

LIST OF ABBREVIATIONS xxi

CHAPTERS

1 INTRODUCTION 1

1.1 Scope 2

1.2 Outline 3

2 LITERATURE REVIEW 5

2.1 Related Work 5

2.1.1 Reflection and Illumination 7

2.1.2 Artifact Extraction 12

2.2 Beam Area Definition and Light Source 14

3 CAMERA SPECIFICATIONS, BACKGROUND MATERIAL AND HARDWARE 19

3.1 Universal Camera Properties 19
3.1.1 ISO 19
3.1.2 White Balance 20
3.1.3 Contrast 20
3.1.4 Exposure Time 21
3.1.5 Exposure 21
3.1.6 Dynamic Range 23
3.1.7 Sharpness 23
3.2 Hardware Specifications 24
3.2.1 Raspberry Pi 3 27
3.2.2 Nvidia JETSON TX2 28
3.3 Camera Specifications 31
3.3.1 Raspberry Pi Camera Module 31
3.3.2 NVidia JETSON Camera Module 33
3.3.3 Other Cameras 34
3.4 Computer Specifications 35
4 SIMULATION AND IMPLEMENTATION DETAILS WITH RESULTS 37
4.1 Setup Specifications 38
4.2 Data Sets 42
4.3 Semantic Segmentation Quality Metrics 45
4.4 Low-Cost System 49
4.4.1 Principles of the Low-Cost System 49
4.4.2 Implementation of Low-Cost System 51

4.4.2.1 Image Acquisition 52
4.4.2.2 Algorithm and Hardware Results . . . 64
4.4.2.3 Temporal Aliasing Problem 73
4.4.2.4 Conclusion 74
4.4.3 Simulation Results for Low-Cost System Algorithm 76
4.4.3.1 Timing Results 77
4.4.3.2 Detection Results 78
4.4.3.3 Evaluation 82
4.5 Mid-End System 83
4.5.1 Principles of the Mid-End System 83
4.5.2 Implementation of Mid-End System 95
4.5.2.1 Image Acquisition 95
4.5.2.2 Algorithm and Hardware Results . . . 97
4.5.2.3 Conclusion 110
4.5.3 Simulation Results for Mid-End System Algorithm 111
4.5.3.1 Timing Results 111
4.5.3.2 Detection Results 113
4.5.3.3 Evaluation 118
5 CONCLUSION AND FUTURE WORK 121
5.1 Conclusion 121
5.2 Future Work 122
REFERENCES 127

APPENDICES

A AUXILIARY TOOLS 139

A.1 Some Methods Commonly Used in Image Processing 139

A.1.1 Canny Edge Detector 139

A.1.2 Otsu's Threshold Method 141

A.1.3 Morphological Operations 142

A.1.4 Histogram Equalization 144

A.1.5 Gaussian Filtering 147

A.2 Hough Transform 148

A.2.1 Variation of Hough Transform 150

LIST OF TABLES

TABLES

Table 3.1 A simple example of rolling shutter and exposure time of 2 cycles . 22

Table 3.2 Raspberry Pi 3 Model B and Raspberry Pi Model A technical specs [1] [2] 28

Table 3.3 Nvidia Jetson TX2 technical specs [3] 30

Table 3.4 Dual power modes of Jetson TX2 [4] 31

Table 4.1 List of sensor modes for V2.1 Raspberry Pi camera [5] 54

Table 4.2 Camera attribute values set manually (for low-cost method and data set) 62

Table 4.3 Time measurements for previewing BGR and YUV raw images . . . 64

Table 4.4 Implemented parameters of low-cost system algorithm variations . . 75

Table 4.5 Average hardware real-time measurements of low-cost system [sec] 76

Table 4.6 Average MATLAB modeling measurements of low-cost system [sec] 78

Table 4.7 Number of convolution filters per tensor channel in blocks 98

Table 4.8 Feature Depth effect on FPS 110

Table 4.9 The time ticks used for each time measurement 111

Table 4.10 Timing results for deep neural network models (Hard Set, batch size:8) 112

Table 4.11 Semantic metric results for deep neural network models (Hard Set, batch size:8) 115

Table 4.12 Two chosen models for hardware experiments 119

Table 4.13 File Sizes of our networks 119

Table 4.14 Fine-tuning of filters in run time on trained models 120

LIST OF FIGURES

FIGURES

Figure 1.1 Some pictures from factories 3

Figure 2.1 Spectral response of human eye, VidiCon and CCD [6] 7

Figure 2.2 Close-up scratch visual on a metal sheet sample 8

Figure 2.3 Presentation of diffuse reflection [9] 9

Figure 2.4 Some configurations for illumination techniques [7] 10

Figure 2.5 Importance of light configurations [8] 11

Figure 2.6 Some lighting configurations from scratch detection literature . . . 12

Figure 2.7 The specular surface under lights on and off 15

Figure 2.8 Two same size light sources with cool white (a) and warm white (b) color temperatures 16

Figure 2.9 Same scene under linearly polarized and non-polarized conditions . 17

Figure 3.1 Depiction of acutance [11] 24

Figure 3.2 NVidia GTX1080Ti performances displayed in log scale. [12] . . . 25

Figure 3.3 NVidia JETSON TX1 performances displayed in log scale. [12] . . 26

Figure 3.4 RaspberryPi 3 performances displayed in log scale. [12] 26

Figure 3.5 Raspberry Pi 3 Model B ports and connectors [13] 27

Figure 3.6 JETSON TX2 Board (*left*) and development kit (*right*) [14] 29

Figure 3.7 The Raspberry Pi Camera Module attached to a Raspberry Pi Board [15] 32

Figure 3.8 Jetson TX2 DevKit Camera Module attached to main board [4] . . 34

Figure 4.1 Early setup for 22.5 cm x 32 cm metal samples 38

Figure 4.2 The latter setup for larger metal sheet samples (69.5 cm x 79.5 cm) 39

Figure 4.3 Explanation of later setup and visualization of beam area 40

Figure 4.4 Schematic of emitter collector relay switch circuit 41

Figure 4.5 Relay switching circuit prototype 42

Figure 4.6 Samples from data sets 43

Figure 4.7 Ground truth for input images given in Fig. 4.6a and b, respectively 44

Figure 4.8 Demonstration of data set separation for 4-fold cross-validation [16] 49

Figure 4.9 Main method of low-cost system algorithm 51

Figure 4.10 Proper connection to CSI port with correct direction 53

Figure 4.11 Effect of brightness (values = 5, 50) 55

Figure 4.12 Effect of contrast (values = 50, 0) 56

Figure 4.13 Effect of metering mode (values = spot, backlit) 58

Figure 4.14 Effect of ISO (values = 800, 200 and 100) 59

Figure 4.15 Effect of sharpness (values = 0, 100) 60

Figure 4.16 Effect of shutter speed on moving image (values = auto(slow), 1000(fastest) and 2500) 61

Figure 4.17 Raw image capture and detection of beam area 65

Figure 4.18 Difference in Histogram Equalization Methods 67

Figure 4.19 Canny Edge Filter results with and without Gaussian blur ($7 \times 3\ kernel$) 69

Figure 4.20 Applying image opening with 3×1 kernel 70

Figure 4.21 A resulting image with its input image 71

Figure 4.22 Low-cost method hardware implementation final results of the main method 73

Figure 4.23 Temporal Aliasing Problem 74

Figure 4.24 Demonstration of weak and strong lines 77

Figure 4.25 Resulting images for low-cost system modeling 82

Figure 4.26 A layered network built by neurons 84

Figure 4.27 A simple deep network with weights indicated on 87

Figure 4.28 Convolution is applied on the entire image. 89

Figure 4.29 A neuron representing the filter W applied on pixels around p_{22}. . 90

Figure 4.30 (a) Image pixel values. (b) Result of max pooling for each colored region. (c) Result of average pooling for each colored region. 91

Figure 4.31 Bilinear interpolation 92

Figure 4.32 ReLU function 94

Figure 4.33 LReLU function as given in Eq. 4.14(where α is 0.2) 94

Figure 4.34 Operations in a single convolution block 97

Figure 4.35 Whole CNN architecture for mid-end system 98

Figure 4.36 Resulting images for mid-end system hardware implementation (Model A) 104

Figure 4.37 Resulting images for mid-end system hardware implementation (Model B) 108

Figure 4.38 Real-time experiment of model trained with NEU data set on Jetson TX2 110

Figure 4.39 Decrease in loss versus iteration 113

Figure 4.40 Loss versus number of iterations for different feature depths (Hard Set, batch size:8) 114

Figure 4.41 Resulting images for mid-end system modeling (Model A) 117

Figure 4.42 Resulting images for mid-end system modeling (Model B) 118

Figure 5.1 Primitive drawing of matrix of lights idea (numbers may vary) . . . 123

Figure A.1 Non-maximum Suppression [18] 140

Figure A.2 Double Hysteresis Thresholding [18] 141

Figure A.3 Basic morphological operations in image processing [19] 144

Figure A.4 Histogram Equalization with images, CDFs (black lines) and histograms (red bins) [20] 146

Figure A.5 CLAHE histogram clipping [21] 147

Figure A.6 Digital Gaussian Filter with different kernel sizes [22] 147

Figure A.7 Demonstration of line equation parameters [17] 149

Figure A.8 A Hough Transform example applied to an image with 2 lines [17] 150

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
ANN	Artificial Neural Network
RPi	Raspberry Pi Board
NDE	Non Destructive Examination
CCD	Charged Couple Device
GPU	Graphical Processing Unit
CPU	Central Processing Unit
FPS	Frame Per Second
SoC	System on Chip
CSI	Camera Serial Interface
GPIO	General Purpose Input Output
SSH	Secure Shell
SDK	Software Development Kit
HDMI	High-Definition Multimedia Interface
PCIE	Peripheral Component Interconnect Express
CAN	Controller Area Network
UART	Universal Asynchronous Receiver-Transmitter
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
SD	Secure Digital
CMOS	Complementary Metal Oxide Semiconductor
API	Application Programming Interface
DSLR	Digital Single-Lens Reflex
MMAL	Multimedia Abstraction Layer

MIPI	Mobile Industry Processor Interface
DOF	Degree of Freedom
LED	Light Emitting Diode
GUI	Graphical User Interface
ROI	Region of Interest
CNN	Convolutional Neural Network
MUX	Multiplexer
USD	United States Dollars
FPGA	Field-Programmable Gate Array
HT	Hough Transform
ISP	Image Signal Processor
FoV	Field of View
DRC	Dynamic Range Control
HDR	High Dynamic Range
EV	Exposure Value
ISO	International Standards Organization
SNR	Signal-to-Noise Ratio
CLAHE	Contrast Limited Adaptive Histogram Equalization
PNG	Portable Network Graphics
JPG/JPEG	Joint Photographic Experts Group
BMP	Windows Bitmap
CDF	Cumulative Distribution Function
GFLOPS	Giga Floating-Point Operations per Second
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
FD	Feature Depth

CHAPTER 1

INTRODUCTION

Due to increase in demand for production, automation is getting more important, gradually. Fabrication should be faster, more efficient and flawless if producer desires to meet demands. In order to be fast enough, machine force is a necessity compared to human force. Machines are likely to finish a job in shorter time and with less errors, in comparison with a human who does very same task. Also, human is prone to drop in performance, while machines are durable and the energy comes from electricity, essentially. Human workers have needs and require strict regulations to work with. In maintenance cost, machines are usually cheaper. A significant advantage of human workers is multitasking and, in fact, being able to accomplish a wide selection of tasks. There is no machine produced yet to match diversity of abilities of a human being. In this manner, academics and robot industry come in action to solve specific problems of customers as quickly and efficiently as possible.

Production consist of different steps. Those steps may vary from a sector to another sector. Depending on the customer requirements, product type, time and environmental circumstances, issues to be solved also vary. Manufacturing is the resulting step, indeed the realization of design steps. Quality of resulting product is what the customer cares about. Consequently, during actual manufacturing whole process should be careful to prevent errors. Those errors may originate from resources as well. Nevertheless, majority of errors occur in manufacturing steps. In fact, resource production can be seen as another manufacturing. Mainly, a manufacturing job comprises transport, flow on industrial line, rotation, cutting, chemical processes such as polishing, painting, bonding and so on. The steps with movement involved, are much more inclined to produce an error. One common output of errors is *surface defect*. During

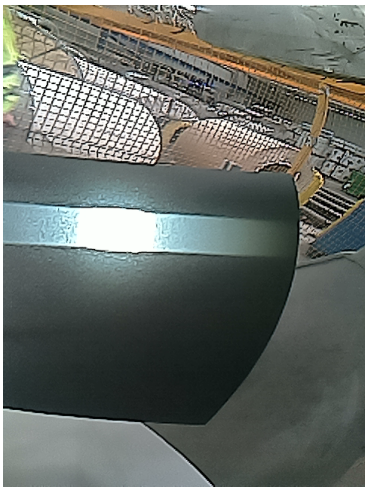
production process, raw material take shape based on the needs. While forming the products degradation may occur on the surfaces. Surfaces can differ from a plain metal panel to a rough fabric, or polished bend wooden plaque. Surface defects are encountered mostly after mechanic operations involving movement, where a major example is cutting [23]. Surface defects take name regarding their shapes and material they are on: dents, bumps, scratches, cracks, stains, discolorations, notches, holes are some examples. Surface inspection is a group methods to examine and detect such defects. Further information on surface inspection and techniques used to realize in literature can be found in Section 2.1. Current literature on surface inspection mainly dominated by the industry. Factories prefer to purchase expensive tools to inspect and test surface materials flowing production line, as it become a hot topic with the beginning of Industry 4.0 trend. There are many important companies that concentrate on such machines. Cognex, Vitronic, Ametek are some examples, and those companies produce robust systems with a multidirectional GUI [24] [25] [26]. Some firms produce inspection systems powered by conventional but expensive cameras [27], in the same way others use advantage of laser technologies [28]. In the end, the production costs brings about high sale prices from [29] starting around some thousand dollars. In some online retail sites, machines starting from approximately 1000 USD to at most 30000 USD [30] are encountered. The capabilities, types of defects they spot, robustness, quickness, size and user interfaces differ.

1.1 Scope

The scope of thesis is to *detect and locate the scratches on metal specular surfaces* considering the requirements given by Arçelik. This work is mainly planned to be used for detection of scratches in the production line of Arçelik Dishwasher Plant. It is going to be a alternative system for current visual inspection, performed by workers (Figure 1.1). The customer requirements that have to be met in the thesis are to a) work in real-time, b) span 1 meter distance in approximately 20 seconds (corresponds to at least spanning *5 cm area at 1 FPS*), c) scan continuous metal coils (namely, *hot-rolled steel strips*) that are around 100 m long and d) detect *vertical line defects*.

The performance measure is defined by how successful the system can detect scratches

(a) Rolling industrial line



(b) An early image acquired by Pi Camera

Figure 1.1: Some pictures from factories

on the metal sheet and locations of those. By answering given needs, it will already solve problems such as "existence of scratches" and "number of scratches".

After comparing techniques given in literature, this thesis work presented one low-cost simple solution with basic hardware (Section 4.4) and one medium-cost (still less costly then large scale industrial systems, see Section 4.5) robust solution with a cutting-edge learning based approach. Proposing an industrial system with its algorithm and hardware included, aiming simplicity, is a new study in this area.

During whole process Windows 10, Ubuntu 16.04, Raspberry Pi 3 and Nvidia Jetson TX2 platforms are utilized. On the other hand, some knowledge on software languages and frameworks like Python, OpenCV, MATLAB, TensorFlow and Unix was required.

1.2 Outline

The document of this thesis study can be examined in the following chapters. After giving an insight about problem, explaining importance of inspection in industry and defining scope in this very first Chapter 1, document moves onto literature review. So, in Chapter 2, current academic works regarding the problem, are presented. There is

also some work mentioned about similar tasks, but directly related to our problem. The main phenomenon, i.e. *reflection* is explained in sufficient detail. Next Chapter 3 is about setup specification, why the hardware used is selected, features of the given hardware and cameras. There is also some important background information on universal camera properties. In the main Chapter 4, data set is presented with performance measurement tools used. Section 4.4 begins with image acquisition details of Hough Transform based method and concludes with actual implementation steps with visuals included. Section 4.5, includes the same sequence for the second hardware method, which is Deep Learning based. Section 4.5.3 and 4.4.3 are about MATLAB and TensorFlow modeling of selected algorithms, followed by discussion on them. The document is concluded with Chapter 5, which gives observations and idea on future possible work.

CHAPTER 2

LITERATURE REVIEW

This section covers the current studies in the literature, related to artifact detection. The aim here is to briefly explain what kind of approaches are used before. The details about light source and illumination configurations given in literature is discussed.

2.1 Related Work

The objective of the inspection job can be detection of scratches, dents, bumps, corrosion, fractures, contamination, non-homogeneous finish/paint or discontinuities. The reason of the flaw is completely specific to the situation. Usage errors, transportation or imperfections that occur during cutting, stacking and welding are common. There are several approaches of inspection regarding problem type [31]:

- Visual inspection
- Microscopy
- Machine based inspection
- Liquid/dye penetration
- Magnetic particle inspection
- Eddy current testing
- X-ray/radio graphic inspection
- Ultrasound inspection

Before focusing on a computer aided solution, one should understand what *industrial inspection* is and how it was developed through years. Those examination methods are mainly divided into two categories: destructive and non-destructive. Naturally, NDE type is preferable in order not to interfere with the physical or chemical feature of the product. In the first times, it was completely sufficient to control products by *visual inspection*. This may be applied to any type of defect, as long as controller is qualified. An important downside is that naked eye visual inspection suffers from limits regarding sensibility of defects [32] . This fact brings about the need of machines for examination, aside from their speed advantage. Below in the Figure 2.1,it is presenting average wavelength span can be sensed by human eye and that by two camera types. Vidicon camera was a common image acquisition tool in early times. Television standards are underlying principle in that low-cost camera. CCD, on the other hand, are solid state devices including a sensor matrix inside [33]. Charge-coupling is a one method of voltage extraction [34].

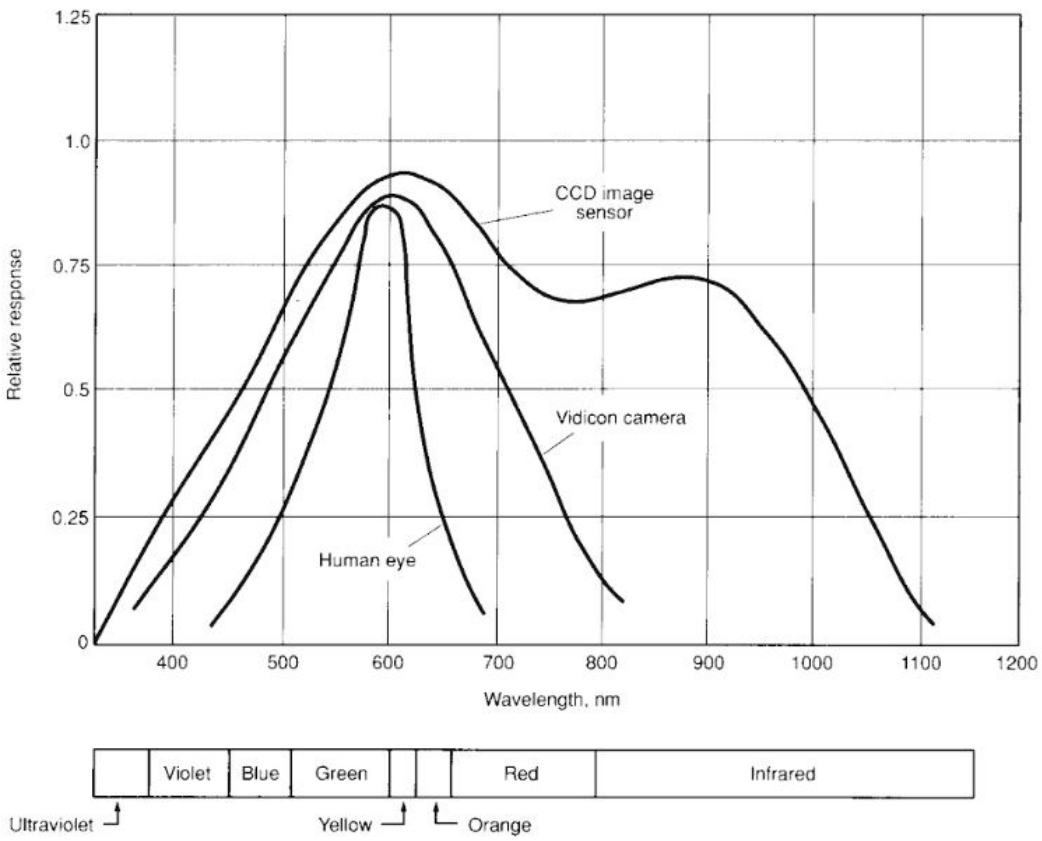


Figure 2.1: Spectral response of human eye, VidiCon and CCD [6]

All in all, human subject inspection is not preferable in many cases due to the facts, yielding subjective results, not being reliable, maintenance cost (salary of qualified inspector), slowness, depending on the vision task, can be not sensible by human (see Figure 2.1) and the truth that repetitive task can be harmful for human subject. Then, it is advantageous to implement a machine based inspection system focuses on the linear type defects such as cracks, scratches.

2.1.1 Reflection and Illumination

Reflection is the main physical phenomenon of this problem. In general reflection, is the change in the direction of a physical wave at the point two different media conjoin. The wave may differ in different environment, water, sound, light or seismic.

However, physical explanation of reflection is similar for different cases.

For our case, reflection of light will be examined. Mainly, it is divided to two nature, namely *specular* and *diffuse* reflections [52]. A mirror exhibits the common sense reflection type that is specular, regular or mirror-like reflection. In this case, glass is coated with a smooth metal surface behind. The basic law of reflection says that incident light reflects from the mirror-like surfaces with the exact same angle, as shown in Figure 2.3a.

If every point incident from an image reflects perfectly on a mirror-like surface, the result would be the exact copy of that image reflected on a screen, if exists. The polished metal surfaces, as in our case, produces such mirror images.

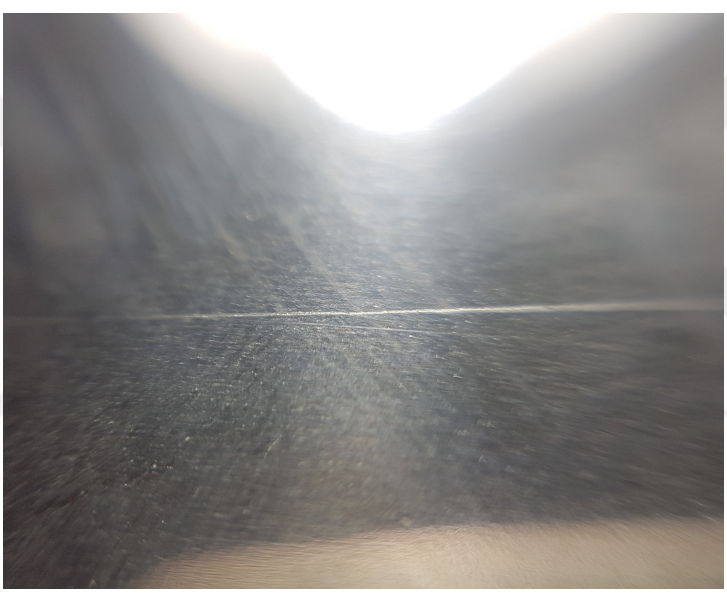
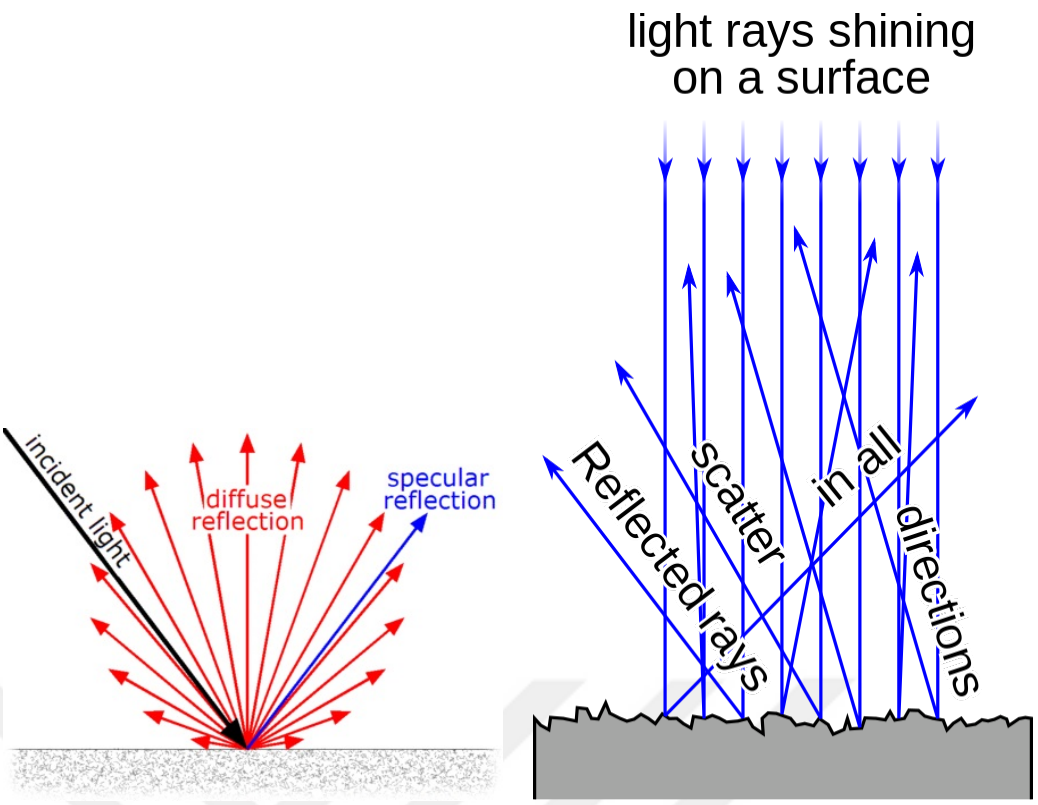


Figure 2.2: Close-up scratch visual on a metal sheet sample

But, the scratches, in contrast with majority of surface area, appear white under certain illumination angle (see also Figure 2.7). This phenomenon is shown in Fig. 2.2b, as well, which is taken using a macro lens. Actually, the reason for white scratches is another reflection type, *diffuse reflection*. In this type, instead of regular ordered reflections (Fig. 2.3a), there observed untidy reflections scattering around due to roughness of the surface (Fig. 2.3b). This is really small *Lambertian Surface* that appears *matte*, in other words has no specular characteristic.



(a) Diffuse and Specular Ref. (b) Diffuse Ref. on Rough Surface

Figure 2.3: Presentation of diffuse reflection [9]

In literature, the illumination techniques are also studied. Figure 2.4 shows some of popular configurations. This configuration totally depends on setup, material and problem to be solved. The importance of those configurations is clearly shown in Figure 2.5 with only three examples. Diffuse illumination reduces glare and provides even illumination, can be obtained with fluorescent bulbs. A ring light is mounted directly to lens and reduces shadowing. It provides uniform illumination when used at proper distances. In polarized light case, illumination is directed that makes use of polarized light to remove specularities and hot spots. This can improve the mirror-effect, but also erase scratches, as well (Fig. 2.9). The choice should be problem specific. Some methods requires extra instruments such as polarizers, ring lights, beam splitters [8].

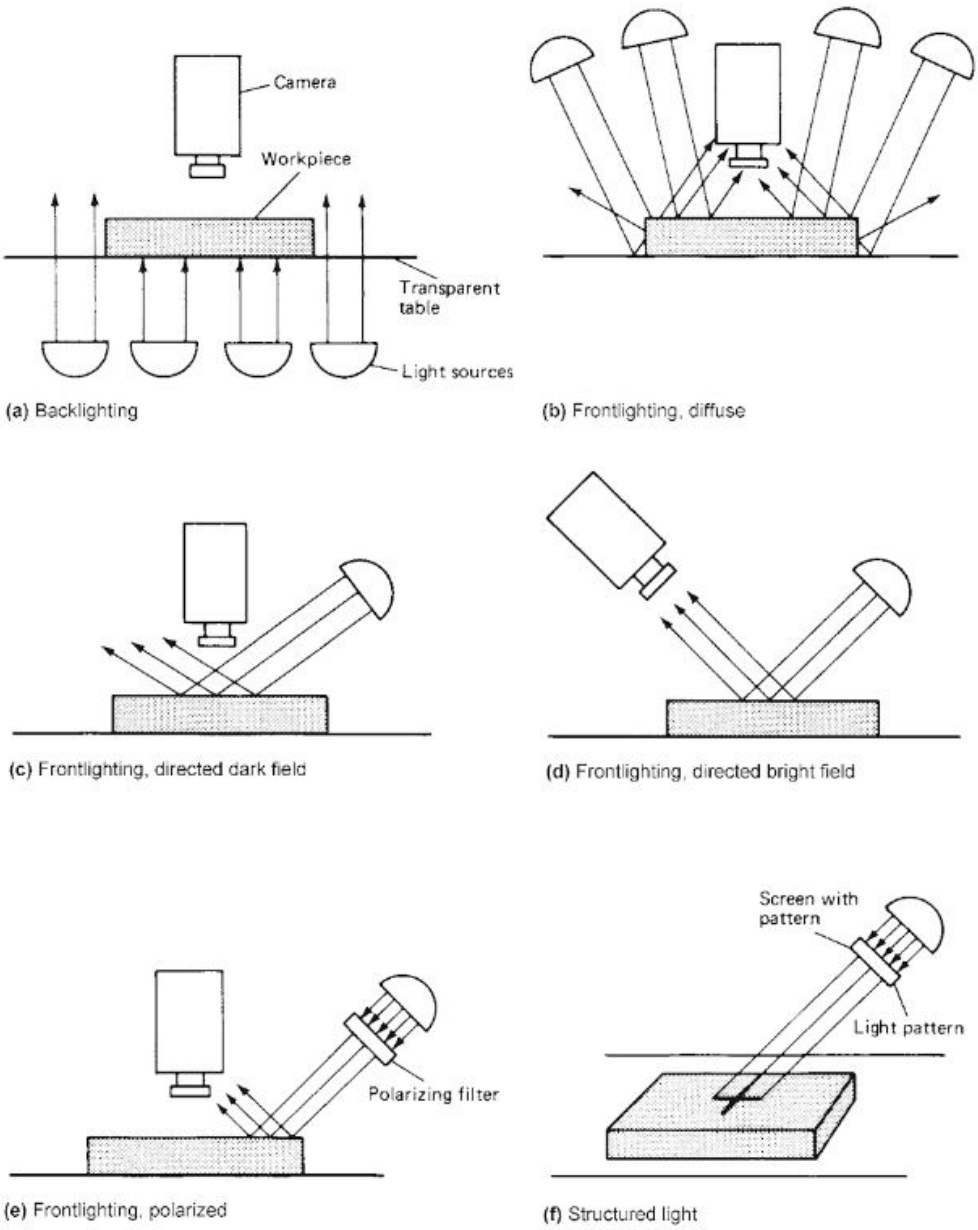
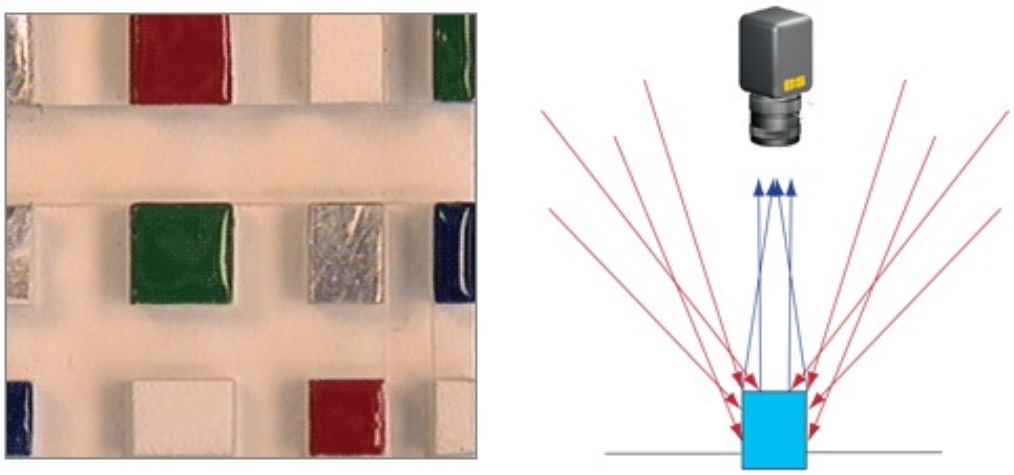
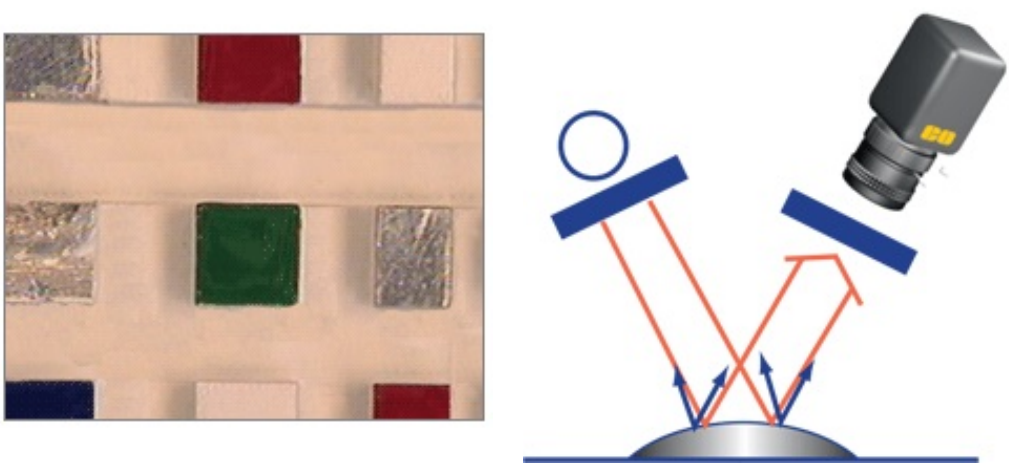


Figure 2.4: Some configurations for illumination techniques [7]



(a) Diffuse Illumination

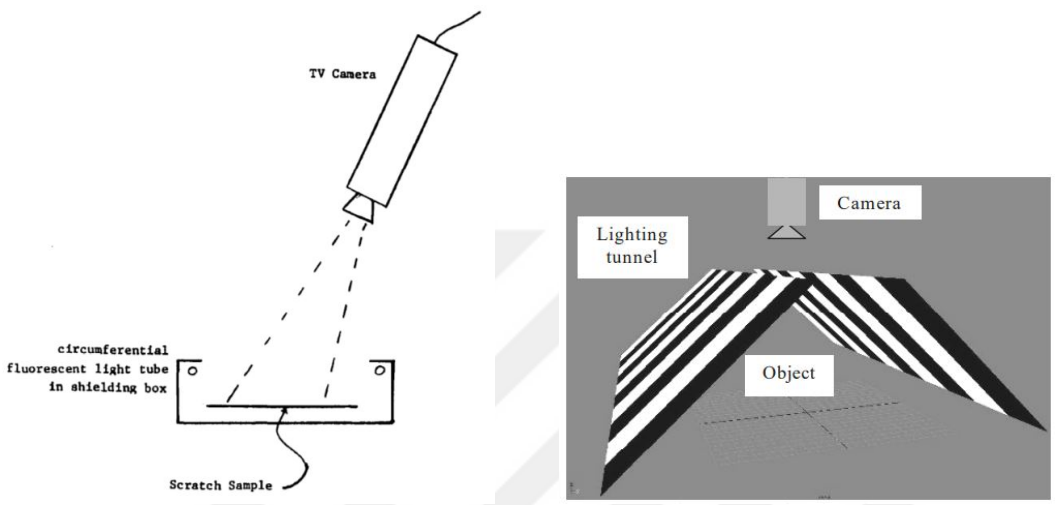
(b) Ring Light



(c) Polarized Light

Figure 2.5: Importance of light configurations [8]

In literature and industry closely related to our problem, there are some common configuration for lighting. For example, in one the earlier studies [50], it is pointed out that camera should view the sample obliquely (Figure 2.6a), the fashion followed in this project as well. Another paper suggested a sliding light tunnel and a camera looking down from top (Fig. 2.6b). As metal product flows constantly in our case, a stable tunnel should be equivalent solution. This one has a more closely covered configuration which is not preferred by our customer, owing to that fact we did not go that path.



(a) Oblique Viewing with TV Camera [50] (b) Light Tunnel with CCD Camera [51]

Figure 2.6: Some lighting configurations from scratch detection literature

Among all those case mentioned above, we would choose *directed bright field*, also known as *direct illumination*, due to *beam area* phenomenon (see Sec. 2.2). Light source leaves a beam trace in the both sides, revealing scratches on this area. In fact, in the final product, it is aimed to create a matrix of lights, so that camera could span all the surface (see switching circuit part in Section 4.1). This application resembles to light tunnel suggested in [51].

2.1.2 Artifact Extraction

Crack detection is a popular subject in different fields of work. It may be detection of cracks in concrete pipes [35] in which it was proposed to use three special

statistical filters. There is another crack detector embedded on a mobile robot to inspect tunnel concrete using Dijkstra Method [36]. In literature crack segmentation are also by morphological operations (A.1.3) aided with Laplacian of Gaussian to mark unqualified solar cells [37]. Those custom methods are usually compared with each other and also some traditional feature extractors like Canny's Edge Detector (A.1.1), Otsu Threshold Method (A.1.2) and popular filters. In this field, there is also another work that is appealing; namely, proposing Gabor Filter to inspect bricks moving on a production line. This problem is closer to ours as it includes a real-time production line. Also Gabor Filter is not prone to fluctuations and mainly enhances crack-like structures which have elongated features [38].

Scratch detection is the main work of field of this thesis work. Although, scratches are one of the most common type of surface degradation, the literature on scratch detection is rather limited. Like any other application, this area is also open for development with emerge of new technologies daily.

Scratch can be created by human error, transportation, mechanics (e.g. cutting, bending), radiation, water and air flow [39]. As is our case, metal surfaces are polished to make it durable and look shiny. For this, the most common method is *chemical mechanical polishing*. In such procedure aluminium is the main component. Scratches occur due to the solid abrasive in particular alumina [40]. There is also the known issue of micro-scratches, that is completely another field, occur due to usually chemical processes. The microscopy comes in action for the solution on polymer, Titanium and Aluminium coated surfaces.For the analysis purposes image processing can be used as well as laser, microwave based systems [41] [42] [43] [44]. Line detection can be done by feature matching [45]. For linear artifact detection for given problem, Gabor filtering is commonly used with high accuracy of 90% [46]. Instead, we have decided to use another common method for line detection on specular surfaces [47], edge detection supported with Hough Transform for its computation efficiency. Recent studies show that [48] [49], artificial neural networks are also successful (both studies are accurate around 98%) on this specific task of surface inspection. Moreover, ANN methods provide robustness.

After detecting that scratch is present, length, width, depth etc. are other features to

analyze. In the scope of this work, we will focus on orientation and location, only. Rest is easier and usually goes parallel with the needs of the customer and surface type, which are not robust.

In conclusion, there are many ways to define features; in our case, mainly linear features. Following feature extraction, there also should be a object detection or segmentation method to spot linear objects. The second part may be tracing, transform or a type of filtering. Whole process probably will include pre- and post-processing to enhance performance in terms of speed and accuracy. At this point, based on hypotheses, some methods will be modeled (see Section 4.5.3, 4.4.3) for performance measuring and eventually hardware implementation will be done.

2.2 Beam Area Definition and Light Source

In Figure 2.7, the mirror like behavior of the metal sheet is shown. This is view of a small sample sent by Arçelik. The image acquired from the actual industrial line (Fig. 1.1b) also gives an idea on how hard it is to visually inspect scratches. During experiments, it was noticed that light reflected on the metal sheet always create a beam of light going both sides (left and right). Furthermore, in this beam area scratches become fairly visible compared to other lighting configuration. Then, we decided use that *beam area phenomenon*, which only appears at *direct illumination*. This illuminated area is owing to the direction of polishing during metal sheet production. So, it was validated by Arçelik engineers that beam always will be in the given orientation.

Illumination is one of the most important aspect in such visual automation project. As a last piece of setup, types of light sources and also configuration of those were studied. At very first glance, when a metal sheet is examined under light, the scratches we are looking for appeared at certain angles, at which metal sheet is held. It was realized that light source itself appears to be so bright in the image, covering any detail on that area (see Figure 2.7). However, the light beam on the both side that falls on the metal surface reveals the details, especially scratches that emits diffuse reflection (see Section 2.1.1).

(a) A view of sample sent from factory (b) Exact same view with lights reflected

Figure 2.7: The specular surface under lights on and off

Due to this, what is needed was a broad light source that will produce a broad light beam area. Among those, it was logical to go for fluorescent or white colors, because we were planning to work with gray scaled images and white light is likely to reveal more contrast around details. There is slight difference between color temperature as seen in Figure 2.8. Here, warm white has less than 3000K, while cool white has

around 6000K color temperature, then it was decided to continue with the latter.

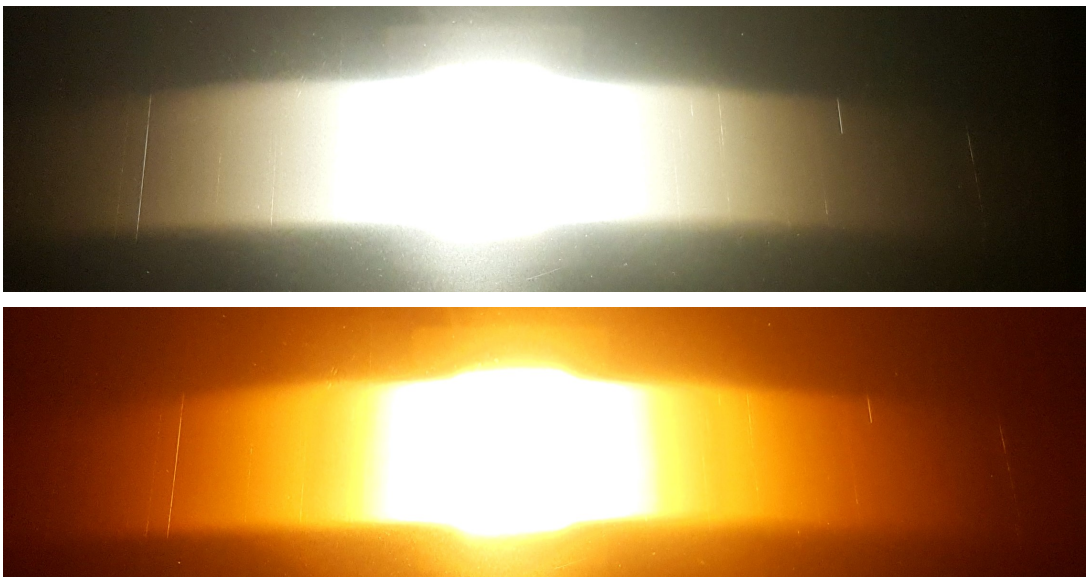


Figure 2.8: Two same size light sources with cool white (a) and warm white (b) color temperatures

After trying small (3 Watts, 4 cm edged square) to medium (6 Watts, 9 cm edged square) square LED panels with different colors, we settled with medium sized 6W cool white LED panel as experiment light source. LED panels has the advantage of uniform illumination, that is desirable for a controlled experiment.

In industrial applications and photography, there is a broad selection of lighting configurations, in Fig. 2.4. Several popular ones are indicated online [8] and can be easily examined. There are setups such as back-lighting, structured lighting, dark-field, diffuse illumination etc. which are out of scope. What we use is direct illumination, but among other choices *ring light* and *polarizer filters* draw attention. Although, there were trials with filters (Fig. 2.9), this was not considered necessary enough. Still, some details are revealed on the light source image area, defects in beam area do not change. Illumination and configuration is a part that is open to variety of solutions (see Future Work Section 5.2).

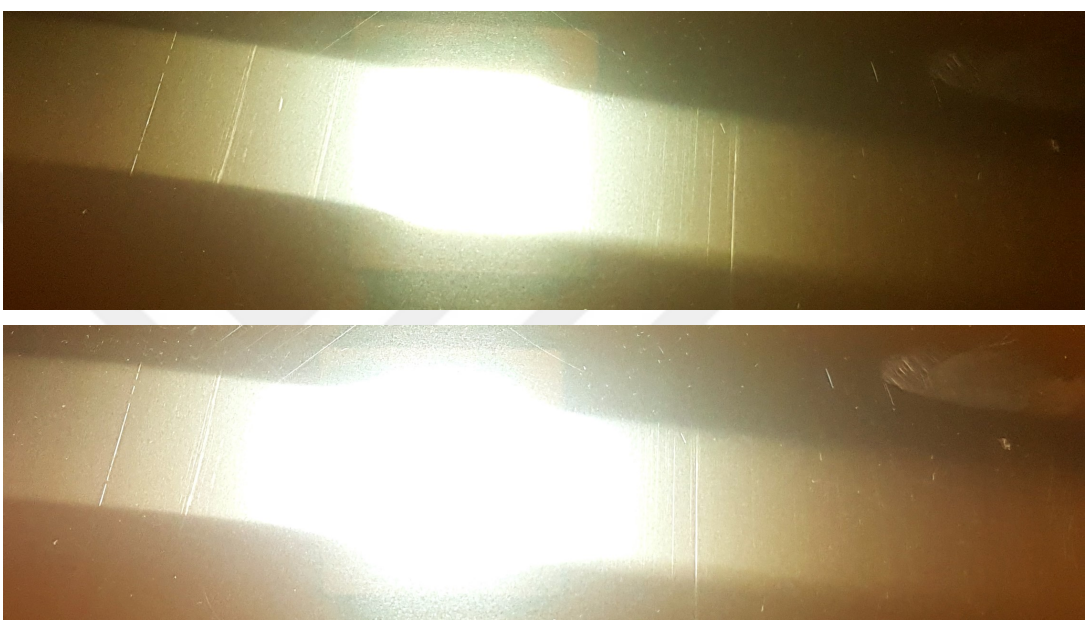


Figure 2.9: Same scene under linearly polarized and non-polarized conditions

CHAPTER 3

CAMERA SPECIFICATIONS, BACKGROUND MATERIAL AND HARDWARE

This chapter focuses on detailed information on background information on used hardware. To start with, the camera parameters used are described in detail. Finally, hardware systems are examined in detail in terms of capabilities and numerical figures.

3.1 Universal Camera Properties

Since the proposed system is based on visual inspection, the camera parameters play an important role in hardware implementation. The aim was to acquire the best image from the camera: fast, conforming to our problem and with enough quality, so that it was a must to study following properties of camera technology. Even though, there are many more important features of a camera that can be adjusted by user, those ,which are related to the problem and can be altered by camera interfaces provided to us, are included in the following subsections. These parameters affected our design and required adjustments are done on the following background knowledge of each.

3.1.1 ISO

ISO, in fact, is a standardization of light sensitivity. Lower values imply less sensitivity meaning less noise, smooth image but in need of better light conditions. High ISO values exposes more noise, but is able give bright details under even low light conditions. The typical values are in between 100 and 1600. Those values are the

arithmetic measure of an older sensitivity metric, ASA [53].

3.1.2 White Balance

White Balance, (alternatively, *Color, Gray, Neutral Balance*) is a photography term about color intensities. Usually adjustments to WB, affects color gains which are used for color conversion during post-process in camera modules. The warmness or coldness can be tuned by photographer by this way to manipulate the feeling of viewers. The white balance of a picture is expressed by Kelvin degrees, similar to the case of external light [54].

There are some mainstream color standards and studies to conform to those accepted standards [55]. To maintain color constancy, there are methods called *chromatic adaptation algorithms*. The task they fulfill is in fact what is know as *auto-white balance*, that is a common function in modern DSLR and mobile phone cameras. Such conversions are made with color conversion matrices (Eqn. 3.1).

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} \frac{255}{R'_w} & 0 & 0 \\ 0 & \frac{255}{G'_w} & 0 \\ 0 & 0 & \frac{255}{B'_w} \end{bmatrix} * \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} \tag{3.1}$$

3.1.3 Contrast

Contrast gives the difference in luminance in relation with average luminance. In general, its formula can be simplified as

$$\frac{Luminance\ difference}{Average\ luminance}$$

This definition can be represented as RMS contrast [56], given by Eqn. 3.2.

$$\sqrt{\frac{1}{MN}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}(I_{ij}-I')^2} \tag{3.2}$$

This definition does not depend on the spatial distribution of intensity, but on the

average of pixel values. I_{ij} is the intensity of a pixel of an image with size of $M \times N$, while I' is average intensity of all pixels in the image.

3.1.4 Exposure Time

Normally, in DSLR cameras, shutter speed (or *exposure time*) simply implies how fast aperture will close after start of a capture. Long exposure time would result in blurry image effects. However as it will be declared in Section 3.3.1, mobile phone cameras have *rolling shutter* technologies, instead. To briefly explain it, consider a sensor array of 3x3 [5]:

In Table 3.1, every sensor cell starts with zero value and frame buffer is empty. Scene is always fully illuminated and cells under light increment the sensor values by 1.

The idea is to reset each horizontal line, consecutively. Then, starting from the first line copying the contents of sensor row into frame row after given exposure time.

In this particular example, exposure time is 2 cycles. The first row resets at instant (b) and get captured at instant (d). Please note that if exposure time was 1 cycle, we would end up with a frame buffer with all ones. In other words, shorter exposure time (faster shutter speed) lowers brightness of the image.

3.1.5 Exposure

Exposure is the total light amount per unit sensor area within exposure time. In other words, it is calculated by aperture, shutter speed and environmental illumination, so there is no direct parameter to control exposure, but indirect ones. Photographers sometimes use exposure and exposure time terms interchangeably. Long exposure may carry the meaning of a shot with long time before shutter closes. The unit EV (i.e. *exposure value*) is calculated by cameras f-number (aperture, camera opening through which light goes in) and shutter speed. EV, when multiplied with scene

Table 3.1: A simple example of rolling shutter and exposure time of 2 cycles

Sensor Elements	Frame # 1
0 0 0	
0 0 0	
0 0 0	

(a)

Sensor Elements	Frame # 1
0 0 0	
1 1 1	
1 1 1	

(b)

Sensor Elements	Frame # 1
1 1 1	
0 0 0	
2 2 2	

(c)

Sensor Elements		Frame # 1
2 2 2	→	2 2 2
1 1 1		
0 0 0		

(d)

Sensor Elements		Frame # 1
0 0 0		2 2 2
2 2 2	→	2 2 2
1 1 1		

(e)

Sensor Elements		Frame # 1
1 1 1		2 2 2
0 0 0		2 2 2
2 2 2	→	2 2 2

(f)

illuminance, again gives exposure [57].

$$Exposure = Sensor\,Unit\,Plane\,Illuminance \times Exposure\,Time\,[lux.sec] \quad (3.3)$$

There is also concept of exposure compensation, which is a kind of an fine tuning forming a feedback for under/over exposed images to become optimally exposed [58]. Most cameras have settings of $\pm$ 3EV, corresponding to 1/3 of f-stops (aperture).

3.1.6 Dynamic Range

In signal processing, dynamic range defines the limitation of a certain quantity, in our case intensity of a pixel or luminance. This is an adjustable capability of many cameras. A higher dynamic range should result in better detail visibility in dark and bright area. Devices with HDR, usually have high contrast ratios, as well [59]. A good example is given on how important is adjusting dynamic range of a device [10].

3.1.7 Sharpness

Sharpness is the combination of resolution and acutance. The latter can be defined as the edge contrast of an image (Fig. 3.1). Perceived sharpness can be stated by the following formula

$$Sharpness = \frac{\triangle Intensity}{\triangle Position}$$

That implies that high resolution images yield less sharpness, while high acutance images will perceived sharper. But, it should be noted that SNR degrades in the same proportion with edge contrast gets greater.

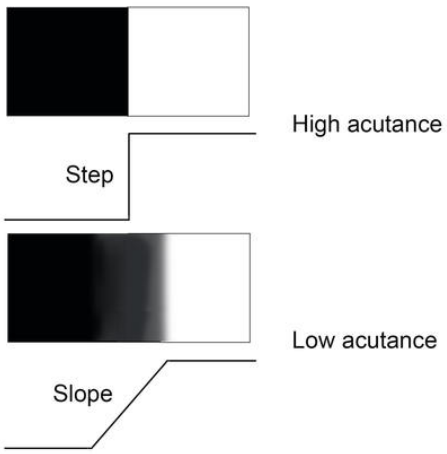


Figure 3.1: Depiction of acutance [11]

3.2 Hardware Specifications

As algorithms take shape and hardware requirements become clear, it was needed to decide on the real-world platform to work with. Nowadays, single board computers such as popular ones RaspberryPi, Arduino, OrangePi, BeagleBone, JETSON, LattePanda and ODROID. Those cards differs in price, CPU, memory amount, intented field of work and popularity [60] [61].

In this context, decision among too many choices was hard, but first filtering was to eliminate those which are not suitable for traditional image processing or machine learning tasks. Those tasks of our scope requires higher parallel processing, memory and possibly a GPU. It was wiser to select two different systems for each task. There are some effort to minimize neural networks or make them suitable to run on CPUs, though [62]. All in all, it seemed more advantageous to implement image and learning jobs on platforms that are intended realize a particular job.

Final step was to make a eventual selection before purchase. At this point, the requirements are over viewed so that:

- Real-time capability should be enough to handle around 3-10 FPS
- Price should be low, as main objective is to propose a low-cost system
- Broad community support, being able to run popular frameworks would be helpful
- System should be easily reachable in our region

Considering above mentioned items, it was decided continue with Raspberry Pi 3 for traditional image processing based system proposal (see Section 4.4). Even though, there were serious rivals such as ODROID in terms of performance, Raspberry's support for frameworks, diversity of community took it forward. Aside from very low-cost system, Nvidia JETSON was the most suitable platform for parallel processing and machine learning solutions [63]. Another consideration was about camera modules, their usability, accessibility and of course, again, cost.

In order to give an insight on why JETSON was seperately chosen for machine learning task, examine Figures 3.2, 3.3 and 3.4.

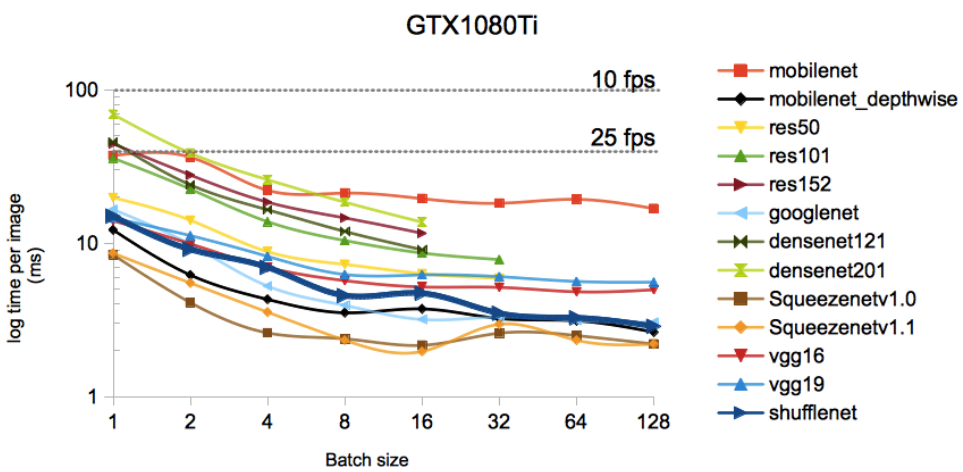


Figure 3.2: NVidia GTX1080Ti performances displayed in log scale. [12]

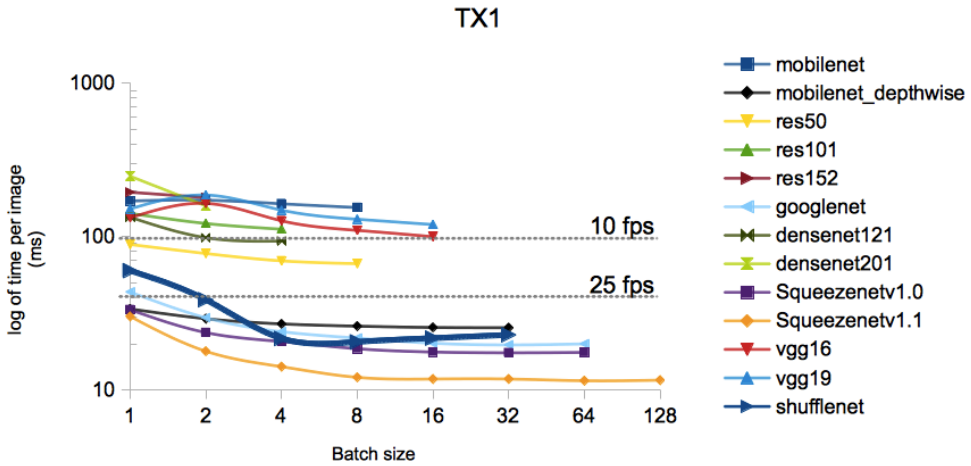


Figure 3.3: NVidia JETSON TX1 performances displayed in log scale. [12]

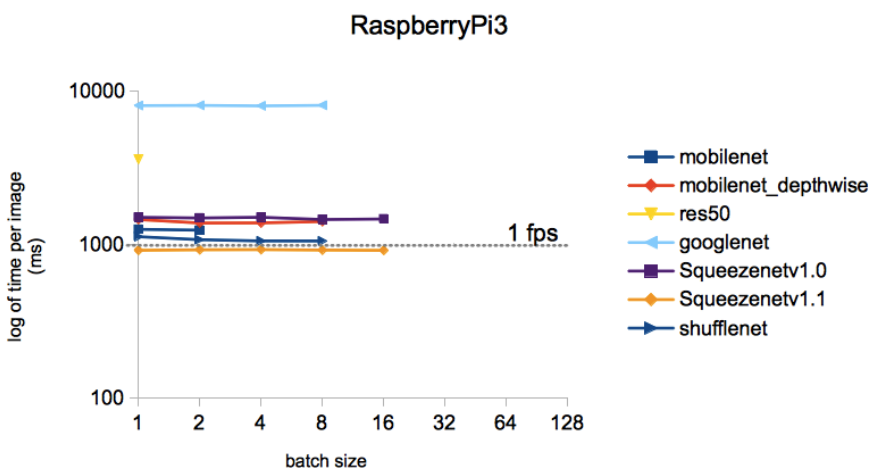


Figure 3.4: RaspberryPi 3 performances displayed in log scale. [12]

Comparing the graphs 3.2 and 3.3, JETSON shows a high performance on DeepDetect Open Source Deep Learning server. Those include some mainstream deep net-

works varying from large ones (*such as VGG*) to smaller ones (*such as MobileNet*). In Figure 3.4, study shows that Raspberry Pi 3 would achieve a real-time detection around 1 FPS, which is not enough for solution of the given problem. Still, a simpler network may be embedded to Raspberry platform (see Section 5.2).

3.2.1 Raspberry Pi 3

Raspberry Pi is an single-board computer, that aims at facilitation of simpler robotic tasks and to make children get used to robotics. It is designed and produced by Raspberry Pi Foundation. Most of the fabrication takes place in Wales, plus in Japan and China [64].

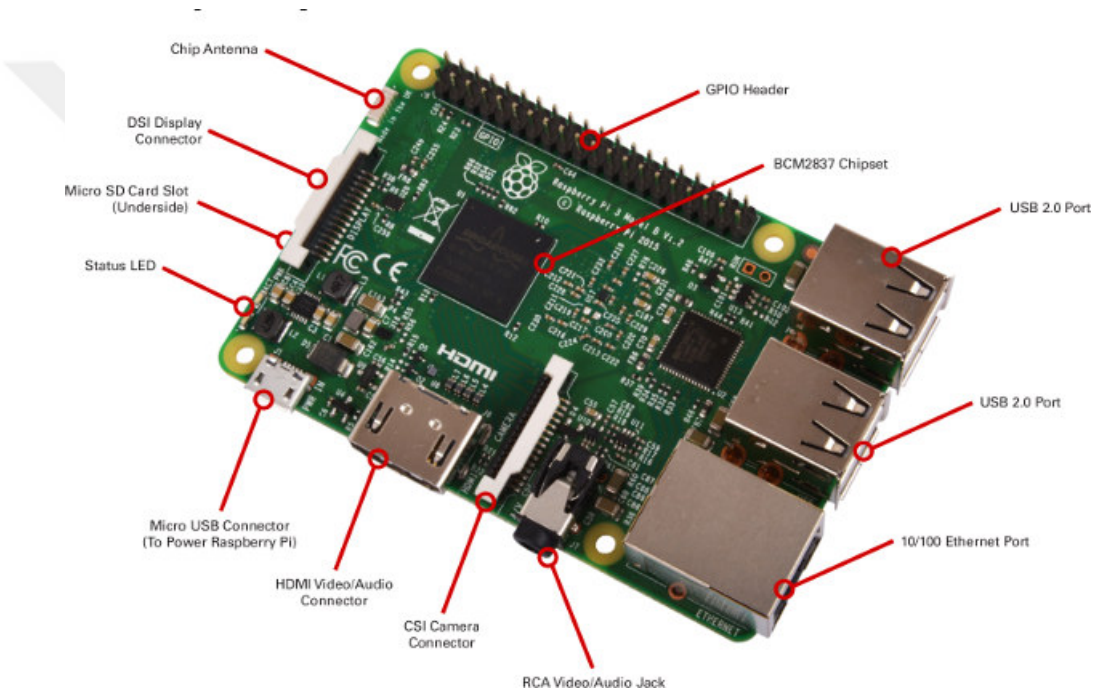


Figure 3.5: Raspberry Pi 3 Model B ports and connectors [13]

Raspberry is a standalone computer, meaning it can be used by hooking up peripherals (*mouse, keyboard and HDMI monitor*) directly. As seen in Fig. 3.5, there are 4 USB ports, GPIO pins, HDMI connector, Ethernet port and CSI camera connector. In Table 3.2, given all technical features that Raspberry Pi 3 and first Raspberry Pi have. Wifi and Bluetooth modules are embedded for Model 3 B. The board is able to handle 1080p H.264 codec at 60 FPS by software decoding [13].

Table 3.2: Raspberry Pi 3 Model B and Raspberry Pi Model A technical specs [1] [2]

	Raspberry Pi 3 Model B	**Raspberry Pi Model B**
Introduction Date	29.02.2016	29.02.2012
SoC	BCM2837	BCM2835
CPU	Quad Cortex A53 @ 1.2GHz	ARM1176 @ 700MHz
Instruction Set	ARMv8-A (64-bit)	ARMv6Z (32-bit)
GPU	400MHz VideoCore IV	250MHz VideoCore
RAM	1GB SDRAM	256MB SDRAM
Storage	micro-SD	SD, MMC
Ethernet	10/100 MBit/s	10/100 MBit/s
Wireless	802.11n / Bluetooth 4.0	-
Video Output	HDMI / Composite	HDMI / Composite
Audio Output	HDMI / Headphone	HDMI / Headphone
Pins (Total)	40	26
Price	$35	$35

The Raspberry card is usually used with an SD card in which installed *Raspbian* operating system. Although, there are many more Unix-like, Windows based OS distributions compatible with the device. It has a very helpful community page and open download source at its official page [2].

In early stages of the project, only Raspberry Pi Model B was provided, so initial test are done on that system. MATLAB has a special tool for Raspberry that can enables user to reach board through network and code via MATLAB interface. After Model 3 B has arrived, we preferred either to connect peripherals directly to the board or working through network via SSH (*network protocol that provides accessing to terminal of remote device*).

3.2.2 Nvidia JETSON TX2

Nvidia Jetson TX2 is a device intended for running parallel-processing application. It is equipped with Nvidia GPU with PASCAL architecture containing a high number of

CUDA cores. PASCAL architecture can be found in all recent cards of this company after April 2016. It is manifactured by 14 to 16 nanometers processes [65]. Jetson series all come with Nvidia Tegra CPU. Each architecture is designed regarding artificial intelligence, computer vision, security and robotics [66]. The main driver of Jetson designs is power-efficiency per parallel processing.

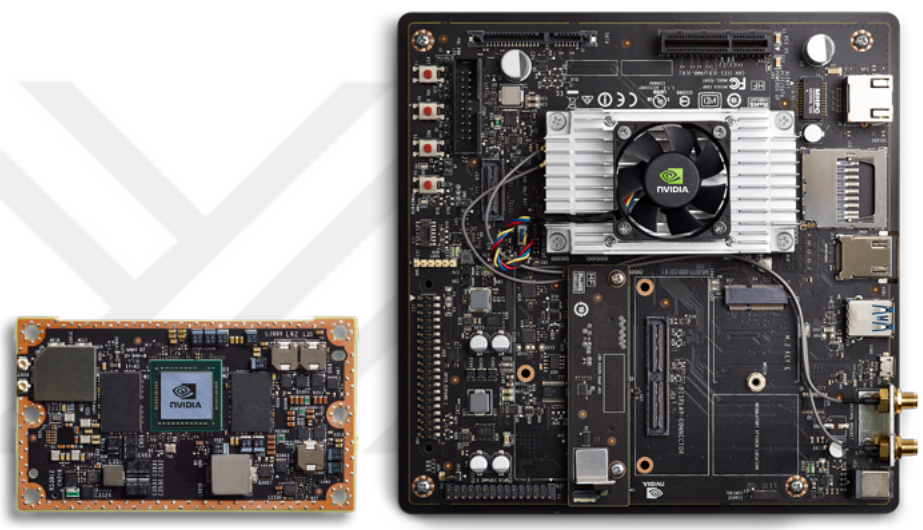


Figure 3.6: JETSON TX2 Board (*left*) and development kit (*right*) [14]

Jetson TX2, which is the latest version of credit size supercomputer, is also available with its hardware development kit 3.6. This includes USB, HDMI, PCI-E, Ethernet, SD, SATA, UART, I2C and SPI interfaces with proper power distribution to units.

Table 3.3: Nvidia Jetson TX2 technical specs [3]

	Jetson TX2
GPU	NVIDIA Pascal™, 256 CUDA cores
CPU	HMP Dual Denver 2/2 MB L2 + Quad ARM® A57/2 MB L2
Video	4K x 2K 60 Hz Encode (HEVC) 4K x 2K 60 Hz Decode (12-Bit Support)
Memory	8 GB 128 bit LPDDR4 59.7 GB/s
Display	2x DSI, 2x DP 1.2 / HDMI 2.0 / eDP 1.4
CSI	Up to 6 Cameras (2 Lane) CSI2 D-PHY 1.2 (2.5 Gbps/Lane)
PCIE	Gen 2 ¦ 1x4 + 1x1 OR 2x1 + 1x2
Storage	32 GB eMMC, SDIO, SATA
Misc	CAN, UART, SPI, I2C, I2S, GPIOs, USB 2.0
Network	1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth
Dimension	50 mm x 87 mm (400-Pin Connector)
Price	$399

Jetson platform has a strong community support. In the same way, it operates with a Linux distribution, able to run popular frameworks such as Tensorflow, cuDNN and OpenCV, that is a huge advantage. It is usual to start with SDK called JetPack.

Jetson TX developer kits add following enhancements [3]:

- USB 3.0 Type A
- USB 2.0 Micro AB (Supports Recovery and Host Mode)
- HDMI
- PCI-E (x4)
- Gigabit Ethernet
- **Camera Module**
- Full-size SD
- Sata Data and Power

- GPIOs, I2C, I2S, SPI
- TTL UART with Flow Control
- Display Expansion Header

For last words, it is a good idea to point out power requirements of Jetson's. There are two new power operations modes which are introduced with TX2. MAX-Q provides maximum efficiency, MAX-P mode promises maximum performance (Table 3.4).

Table 3.4: Dual power modes of Jetson TX2 [4]

	Power Consumption	Note
MAX-Q	7.5 W	Up to 2 x the energy efficiency of TX1
MAX-P	15 W	Up to 2 x the performance of TX1

3.3 Camera Specifications

In the following section, a small detail on the camera specification for each system will be given. Both systems have a common choice of camera on their own. Jetson TX2 development kit comes with that choice, while Raspberry Camera Module should be acquired separately for a fair price.

3.3.1 Raspberry Pi Camera Module

The camera module used in this thesis work is *Raspberry Pi Camera Module v2.1*, to be more precise. It is a low cost grade device with a price around 25 USD. This module is equivalent to more basic mobile phone cameras rather than professional DSLR cameras. The heart of the module, IMX219 sensor, is produced by Sony. The sensor is a 8-megapixel on with CMOS technology. Module is capable of taking 1080p30 and 720p60 footages, also still images [15]. As seen in Figure 3.7, module

is hooked to CSI port of the board via an 15-pin flex ribbon cable. Focus is fixed from reaching by software, to change it a pliers-like tool has to be used firmly.

Figure 3.7: The Raspberry Pi Camera Module attached to a Raspberry Pi Board [15]

There are a lot of examples on how to use the camera in multimedia applications such as time-lapse, slow-motion; but, alternative camera options lack of information on the Internet. What we have used for interface was a third-party Python library (namely, picamera see Section 4.4.2.1), there was not sufficient information either on the official Raspberryi Pi Camera page [15] or library document page.

Sensor IMX219PQ, manufactured by Sony, has some innovative achievements, but also it is suspected that some of these feature affected our object analysis results. One of these features is Lens Shading Correction Function [67]. Although there is no quite detail on the official sensor page, it is function that corrects shades with four independent colors. Anything automatically done is a suspect for the temporal aliasing problem which is introduced in Section 4.4.2.3.

Rolling shutter is the essential principle behind image capture of this camera [5]. In fact, not the whole sensor matrix captures signals at the same time instant, in contrast with *global shutter*. So CMOS sensor horizontally or vertically scan the signals, but a single image is not formed by a single time instance scan. In this manner, camera production cost is lower and also shutter never fully closes (as in a DSLR camera); as a result, camera is more sensitive to changes [68].

The hardware units are explained in highly detailed text that can be considered for further reading [5]. In this page, how every step works is defined and what is the architecture is shown.

Lastly, camera functions through MMAL API. MMAL stands for Broadcom's C library to access video core. This interface has 3 port access, that are *still port*, *video port* and *preview port*. The functions of these will be explained in implementation in Section 4.4.

3.3.2 NVidia JETSON Camera Module

It is full of information online how Raspberry Pi Camera Module works, moreover its sensor and software API. Conversely, it is rather hard to get information about *Jetson TX2 Development Kit Camera Module* (Fig. 3.8). Still, there given specifications and user guides on the camera module.

The specifications of the Jetson Camera can be itemized as:

- 5 Megapixels
- Fixed focus
- MIPI CSI connection

The module is equipped with an Omnivision OV5693 Bayer Sensor [69], which provides 5-megapixel low noise frames in high FPS. The stand-alone price of such a camera module is again $25USD$ (we acquired Jetson TX2 Development Kit for $600USD$, comes with camera module already). Actually, those two cameras are very alike in terms of price pier and performance. Raspberry Pi camera module seems to be

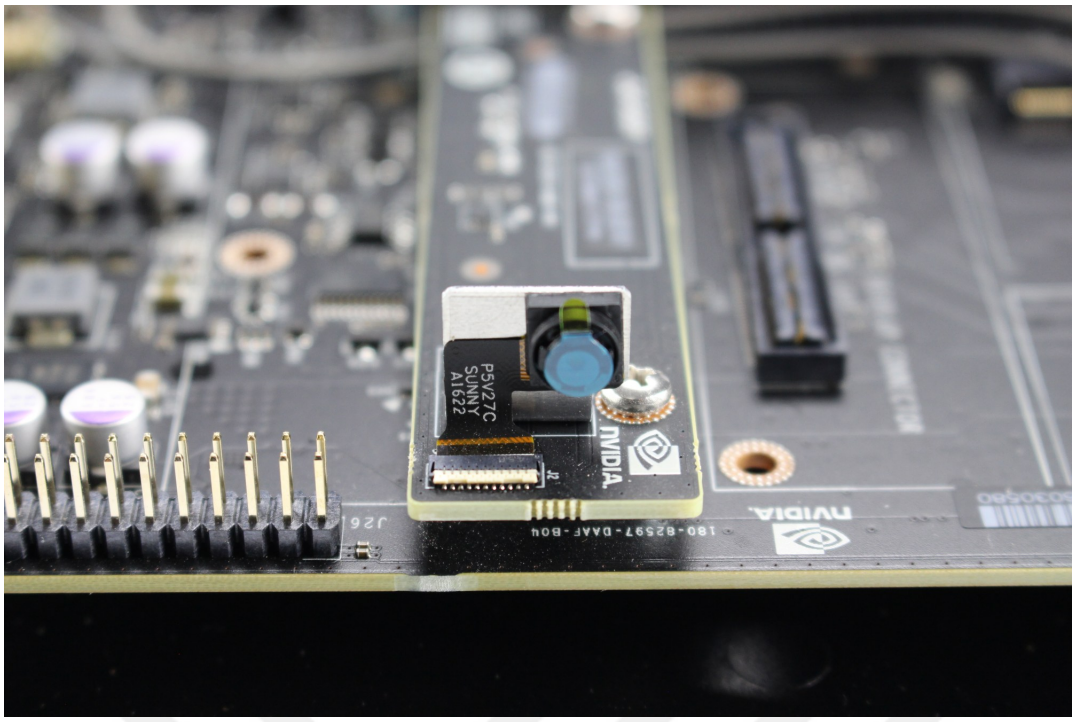


Figure 3.8: Jetson TX2 DevKit Camera Module attached to main board [4]

slightly better than Jetson's, at first glance. But, image processing and transmission power of Jetson is much higher, providing mid-end hardware better capabilities for real-time applications.

3.3.3 Other Cameras

In fact, in such implementation involving a flowing line of product, line-scan cameras are the most popular devices to capture images. They consist of CCD sensor arrays [33] which are linearly aligned each connected to a MUX chip. The camera quailty and speed rate depends on the MUX and CCD tecnologies. The serial analog signal incoming from MUXes are proccessed by a microproccessor to acquire digital data to be processed by a computer [70]. Also, other matrix form CCD cameras could be used as in literature [36], but both types are extremely costly compared our low-cost solution. The prices begins from approximately 300USD on online retail websites.

Regarding deep learning solution, high resolution images are not necessary, in fact a burden for neural networks. Low resolution images are preferable. And line-scan

logic does not fit convolutional network logic, as well. Consequently, it was decided to go with simpler mobile phone grade camera, which are fairly inexpensive.

3.4 Computer Specifications

In this short section, some important specifications and platform information on the computer used for modeling is given. MATLAB modeling is utilized on Windows 10 platform, while TensorFlow framework with Python is used on Ubuntu 16.04 version. Both platforms are installed to a laptop computer with the following system attributes:

CPU Intel Core i7 4710MQ @ 2.50GHz

GPU Nvidia GeForce GTX 980M 8GB GDDR5

GPU2 Intel HD Graphics 4600

Memory 2x8GB DDR3 @ 1800MHz

OS Windows 10 64-bit

OS2 Ubuntu 16.04 64-bit

CHAPTER 4

SIMULATION AND IMPLEMENTATION DETAILS WITH RESULTS

After literature search, it was needed to decide on proposal methods, based on the performances. To set an anchor and to visualize expected final hardware results, a software model for each methods to be implemented on hardware. This is an important step as realization on hardware directly is lot more time-consuming and inefficient for performance comparison.

An important note is that the basis problem is finding linear objects in the image, which is a rather simple problem for object detection and feature extraction literature. So, we needed to scan and filter related work based on robustness, speed, viability, innovation and relation with our specific task. Here, in this chapter we will first model two of those selected methods and compare results in terms of success on the data set with timing. Then, two selected methods will be implemented on hardware in the later sections. Both will be examined in terms of timing and accuracy performance with popular metrics.

At this point, we decided to go from two different routes. One, is a simple way of detection of linear objects in an image, rather less hardware exhausting, namely Hough Transform based pure image processing (Sec. 4.4). The second method (Sec. 4.5) would be, in contrast, lot more complex in terms of number of operations, but resulting in a robust, fast and accurate system. In this point, feature extractors are studied, however, we decided to set a Convolutional Neural Network to be specially trained for our specific task.

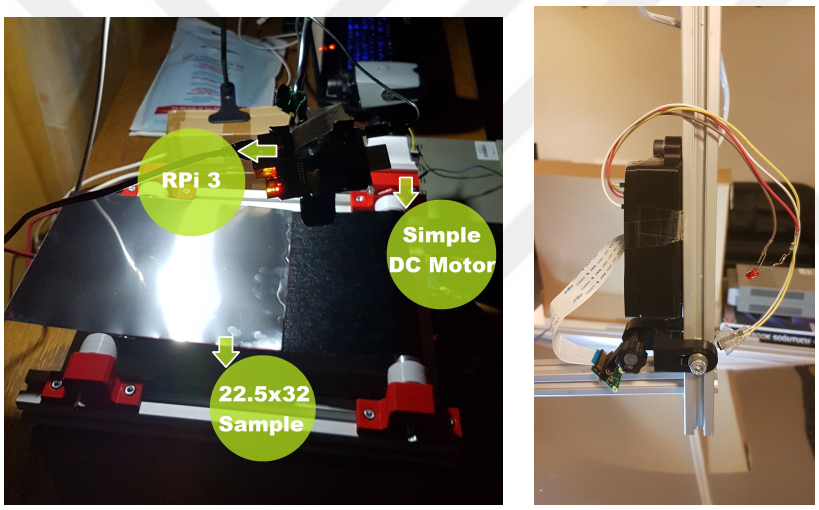
In first sections of this part introduce the details on common performance evaluation methods, metrics and data environment. The chapter also gives details and visuals

of experimental setup units. Mechanical frame was provided by METU Mechanical Engineering Department.

4.1 Setup Specifications

The dimensions and formation of the setup are determined after hardware choices are finalized. Please see Section 3.2 for type of hardware chosen and dimensions of them.

All the laboratory setup frames are designed and manufactured by Mechanical Engineering Department, Middle East Technical University. The processes were led by Assist. Prof. Dr. Ulaş Yaman. The skeleton is realized with sigma profile bars, which are highly versatile, as a result, appropriate for research and development project such as ours.



(a) Small Setup (b) Raspberry Box

Figure 4.1: Early setup for 22.5 cm x 32 cm metal samples

In Figures 4.1, early version of setup is shown. It consist of a small industrial line representation for the former samples which are provided to us. These were 22.5 cm to 32 cm in average. This early setup had also a small box to protect Raspberry Pi with all proper holes are located for input output ports. The camera had an adjustable 1-DOF knob. For simulation and demonstration purposes, a simple DC powered motor is attached to rotary, so line could flow and movement was imitated.

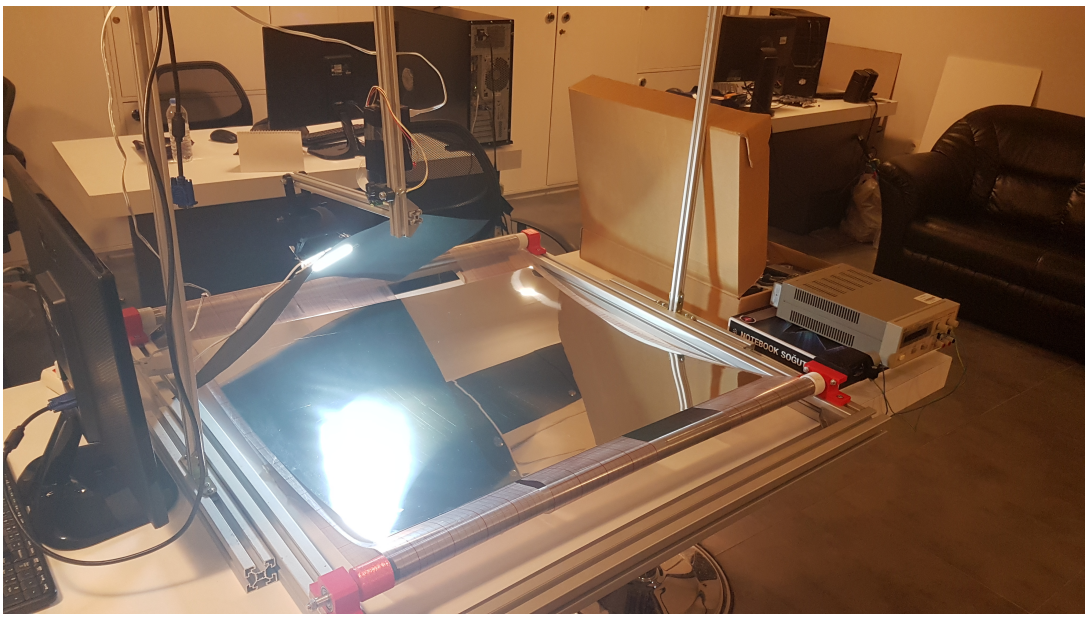
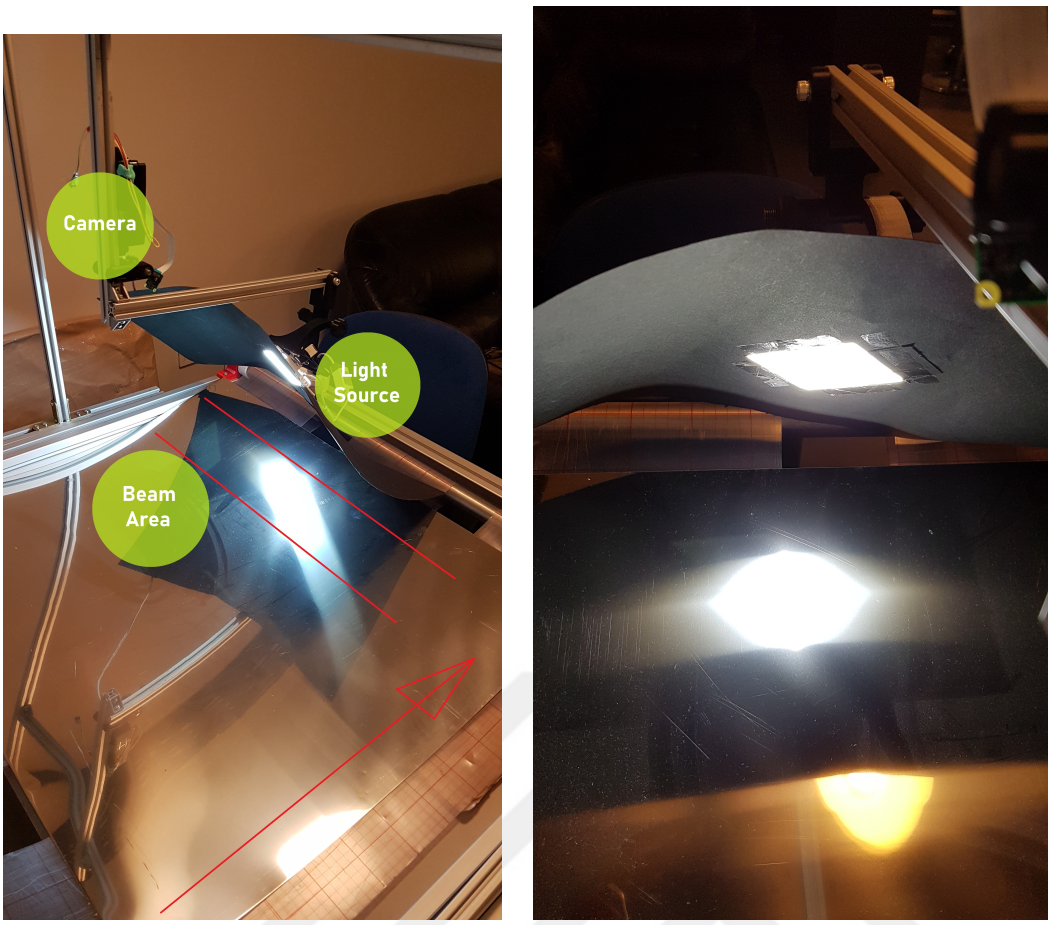


Figure 4.2: The latter setup for larger metal sheet samples (69.5 cm x 79.5 cm)

Figure 4.2 is a photo of final laboratory setup, which can handle larger metal sheet(69.5 cm x 79.5 cm) simulations. The Raspberry Pi box and camera holder stayed the same, frames are added above the industrial line to carry Raspberry box and the new 1-DOF holder of light source. Moreover, DC motor is updated to be able to run heavier weight. The light source was carried by a flexible phone/tablet holder in the first setup. In Figure 4.3a, another angle is presented to show later setup. The metal sheet flows in the direction shown with red arrow. Camera (Raspberry Pi or Jetson) sees the beam area (also in Fig. 4.3b) between two red lines. The ROI image falls into a dark background, due to black cardboard attached to the light source. The details of light source selection and illumination configuration is given in Section 2.2.

Switch Circuit is a relay circuit that enables a digital signal to control AC power to reach light source. This was needed in the final product (as stated earlier in this section) to create a matrix of light sources. By doing so, it will be possible to span all metal surface, as light source itself appears very bright in images and hides details in that particular area.



(a) Units of Setup (b) Camera Point of View

Figure 4.3: Explanation of later setup and visualization of beam area

The aim was to control AC power distribution to light sources by Raspberry Pi (or Jetson) GPIO pins. In that manner, following simple circuit is prepared, which is called *Emitter Follower Relay Switch Circuit* in electronic literature [71] (Fig. 4.4).

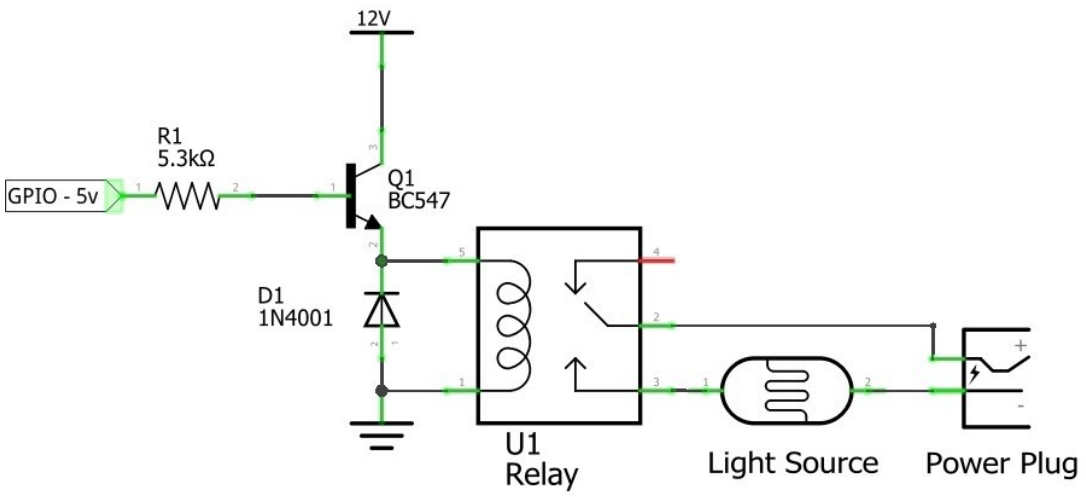
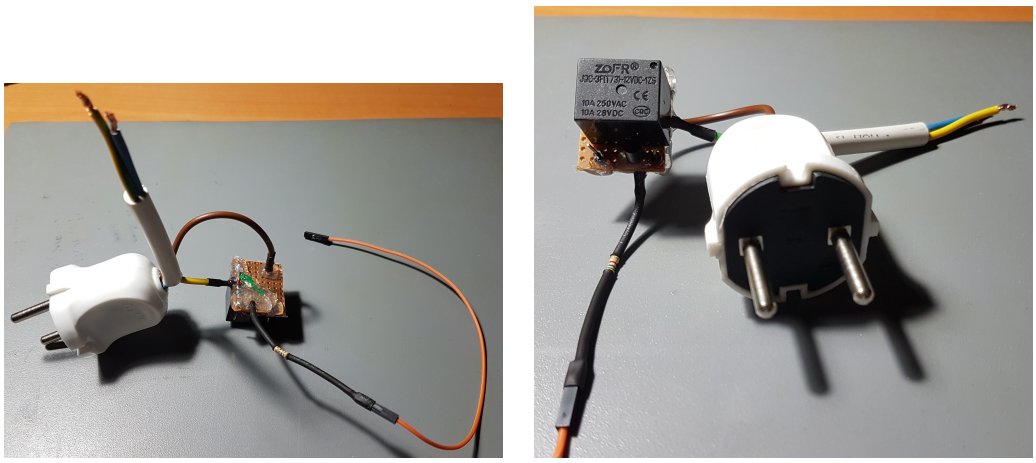


Figure 4.4: Schematic of emitter collector relay switch circuit

Relay is the main component of the circuit. It channels signal at COM port (in this case, plus of electric plug is connected) to NO (normally open) pin whenever coil is energized; otherwise it channels COM to NC (normally closed) pin. Coil power is controlled by basic transistor switch logic, in which base is supplied with sufficient positive voltage to let current flow through collector to emitter. In our case, base voltage comes from Raspberry Pi (or Jetson) GPIO through a resistor to limit current. When transistor is open, 12 volts VCC (collector voltage) reaches positive of relay coil. Real prototype is homemade and shown in Figure 4.5.



(a) Plug, light source cables, diode, GPIO cable(red)

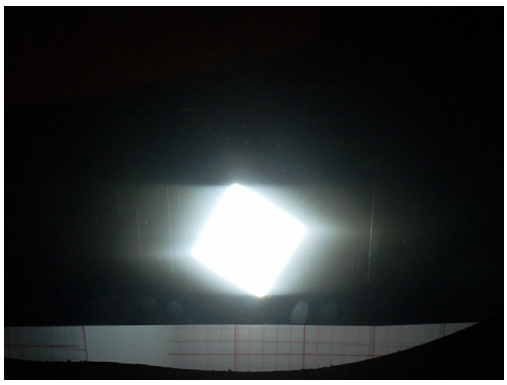
(b) Relay, transistor

Figure 4.5: Relay switching circuit prototype

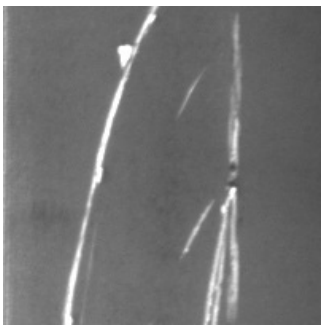
There is also a diode functions as a reverse current protector. This common diode is called *flywheel diode* in this case. The reverse current occurs due to fact that relay powers up by a coil, and it creates a current to release the stored energy. This reverse current may be high enough to kill switches or transistors [71].

4.2 Data Sets

In this thesis work, there are 3 data sets were generated in total and 1 is downloaded from the Internet(Fig. 4.6). Two of those earlier data sets are disposed due to unstability between sample images or confusing objects in the view, though. Of course, real-time hardware applications are not suitable for data set experiments. Modeling sections (Sec. 4.5.3, 4.4.3) is the main scope for data set use to compare methods in a common ground.



(a) Dataset 1 (Hard dataset)



(b) Dataset 2 (NEU dataset)

Figure 4.6: Samples from data sets

One of the data set is acquired from literature (Fig. 4.6b). The authors have created a 1800 image data set for 6 types of surface defects [72]. They implemented this set in their article work [73]. In this defect data base, there are only 300 of 200×200 pixels scratch images with ground truth provided as rectangles. This data set created in Northeastern University, will be referred as *NEU Dataset* from this point forward. The problem with this set is that images are taken partly using microscopy. So the scratches appears thick compared to our intended system, but this set is also included in testing proposed methods. Ground truths are marked as white bounding boxes, see Figure 4.7b.

The second data set is prepared in our laboratory. Raspberry Pi Camera Module v2.1 was used for that purpose. All the camera settings are stated in Sec. 4.4.2.1 (see Table 4.2) and general camera hardware specifications can be found in Sec. 3.3.1. Likewise, ground truth for this study is also prepared as rectangles (Fig. 4.7a). The samples were provided by Arçelik Dishwasher Plant:

Small Metal Sheets 8 pieces (each app. $22.5 \times 32cm$)

Large Metal Sheets 3 pieces (each app. $69.5 \times 79.5cm$)

There are 500 sample images in Hard Dataset (Dataset 1). To acquire that many images, all large metal sheets are put on flowing product line in laboratory. Approximately 50 frames are captured for a slow flowing case (app. 0.5 cm/sec), then sheets

are rotated 180 degrees for another 50 frames. In addition, for each metal sheet, a faster experiment is done to acquire more blurry images. There were initially 600 image samples taken, but 100 of them were eliminated, because they had incorrect sights of mechanical parts. After acquiring images on Raspberry Pi, the rectangles are drawn by hand one by one with the help of integrated MATLAB Sample Labeler Tool.

The dataset is acquired by a basic function block written for this purpose. Whenever the hardware button is activated, Raspberry Pi creates a new folder, named by time information. A sequence of 100 images are stored in there, an LED informs user that procedure in finished. The reason this dataset is called *hard set* is the fact that images are taken under low light condition and scratches are proportionally thinner, in comparison with other datasets. Consequently, they became harder to spot, but closer to the final system in our mind.

(a) Dataset 1 (Hard set) (b) Dataset 2 (NEU set)

Figure 4.7: Ground truth for input images given in Fig. 4.6a and b, respectively

Other datasets were given up because images were not with enough quality. The motion blurring was a huge problem for moving cases, which is eliminated in hard dataset after studying Raspberry Pi camera profoundly. Also, light source was smaller (3W) and changed to 6W alternative in hard dataset. In fact, in the earliest dataset, the scratches are close up shot and appear very clear. The problem was the holder cables around and the angle of the light source, of which edges sometimes could be mixed with real linear objects. Those problems are solved in *hard dataset*, by installing a

black cardboard behind the lamp and rotating the LED panel by 45° ,so that its sides would not cause vertical lines in the edge maps.

Finally, Hard Dataset including 400 training, 108 test images and NEU set including 200 training, 100 test images are prepared, as explained above.

4.3 Semantic Segmentation Quality Metrics

A pixel-wise segmentation of an image requires some common ground evaluation for comparison of performance on how accurately the method can detect different classes. For this purpose, there are some global metrics defined by pixel count calculations. In this study, we will consider some of the standard, however, there are still new metrics are emerging depending on needs. For example, this paper proposes a new image segmentation metric based on class boundaries [74].

Besides those digitized methods, there are also approaches for *regression* problems, which are problems with probabilistic results not binaries.

For pixel-wise logic operations, some most popular ones are shortly described below. Further information on expansion to those may be found in [75]. In order to compute following metrics, there we need a ground truth and a resulting matrix (or an image) from our algorithm, in binary form. As we have only one class of objects (scratches), we can talk about binary types. More classes would bring about complex *confusion matrices*, that is formed by *true positive, true negative, false positive* and *false negative* entries.

True positive (*hit*) pixels represents correct hits of segmentation of the given class. In other words, this is the count of pixels that is mark in ground truth matrix and output matrix of method to be evaluated.

True negative (*correct rejection*) is in the reverse side, being correct prediction of non-object pixels. This gives the count of background pixels in ground truth, that are not object pixels in the resulting matrix, either.

Moving onto the erroneous parameters, **false positive** (*Type I Error*) pixels are estimated as objects in the resulting image, actually they are not.

In the same way, the pixels that are predicted as background after algorithm calculation and originally belong to an object are called **false negative** pixels (*Type II Error*).

Depending on those basic definitions, there are some further parameters coined in this context. Some of globally used advance parameters are:

- Sensitivity
- Specificity
- Confusion matrices
- Precision
- Miss-rate
- Accuracy
- F_1 score
- Markedness
- Informedness
- Matthews correlation

  etc.

Sensitivity

This parametric measure is also called *recall, true positive rate* or *probability of detection* in some fields. This is the most popular ratio with specificity and given by following equation (Eqn. 4.1. It expresses proportion of correct hits in all object pixel predictions. A high sensitivity can be equal to correct hits on objects but with a poor precise localization.

$$Sensitivity = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{4.1}$$

Specificity

This one has also alternative names like *true negative rate* or *selectivity*. Its simple mathematical formula is in the equation below (Eqn. 4.2). As specificity goes higher, it means that in all background image, correct background prediction is done.

$$Specificity = \frac{True\ Negative}{True\ Negative + False\ Positive} \tag{4.2}$$

Accuracy

Accuracy is another metric widely used in performance measures of image segmenters. This has basic logic behind, that is all correct predictions in whole image, given by Eqn. 4.3.

$$Accuracy = \frac{TN + TP}{TP + TN + FP + FN} \tag{4.3}$$

Cross Entropy Loss

Cross Entropy Loss differs from the metrics above by that it suits for regression problems, that results with probabilities in contrast to discrete outputs. One of this kind of metric is needed, because deep learning network model returns soft prediction results for each class (one class in our case) and feeding back soft values results in better optimization.

For this task, some famous regression metrics like *mean squared error* (MSE, L2) or *mean absolute error* (MAE, L1) could be preferred. But, for image segmentation tasks there are other loss functions implemented usually. Cross entropy loss is one kind, Huber, Hinge and Dice loss are the other examples. It is given by function definition in Equation 4.4 [76].

$$D_{loss}(S, L) = -\sum_{i}^{C} L_i log S_i - \sum_{i}^{C} (1 - L_i) log(1 - S_i) \tag{4.4}$$

In Eqn. 4.4, S represents soft output of neural network and L comes from the class membership from ground truth. The left summation represents correct hit on a pixel belonging to a class, meanwhile, right side is for background predictions. The number of classes C on the summation is 1 for our case, so formula can be simplified further.

Please note that, from this point on, cross entropy loss will referred as *loss* and only will be used for mid-end learning based model.

10-fold Cross Correlation

To obtain stable and correct results of segmentation metrics, there are some common validation methods. The idea behind those is training a model with different clusters of train and test data sets, and covering all data set to reduce biased results. *Leave-p-out cross-validation* and *k-fold cross-validation* are most popular iteration of those method [16]. 10-fold validation was chosen in this study, due to its fair evaluation performance [77].

First, all data set is shuffled and this order is fixed for all models and also Hough Transform based method, to be equivalent for all performance measures. Then, data set is divided into 10 parts (each having 100 samples) and models are trained with remaining 400 samples in each case. An example for 4-fold validation is depicted in Figure 4.8. After all iterations (k=10, for our case), the average is taken for each semantic metric.

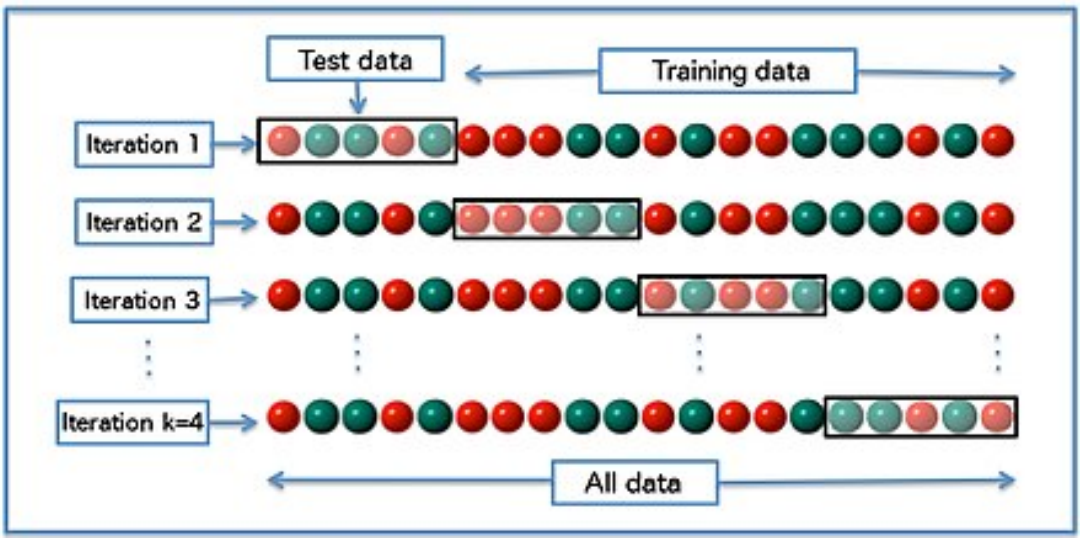


Figure 4.8: Demonstration of data set separation for 4-fold cross-validation [16]

4.4 Low-Cost System

4.4.1 Principles of the Low-Cost System

The main artifact extractor of the following implementation is called *Hough Transform* (App. A.2), which is a rather old (1962) method to identify linear features in an image efficiently [78]. There are also pre-processing tools used to enable successful feature extraction, for which explanations could be found in Appendix part A.

Here is the pseudo-code showing how main function works in the embedded system:

Algorithm 1 Algorithm for Low-Cost Method

```
Library Definitions
procedure MAIN
    LED Pin Definition
    procedure WITH CAMERA SETUP(sensor_mode, resolution, framerate)
        CameraAttributes ← CorrespondingValues
        Sleep : 2sec
        for frame in CaptureMethod(format, MMAL_port) do
            < Time Tick 4
            < Time Tick 1
            Main Algorithm                          ▷ (see flowchart Fig. 4.9)
            < Time Tick 2
            Result Showing                          ▷ Optional
            < Time Tick 3
        end for
    end procedure
end procedure
```

Embedded code (written in Python), starts with including necessary libraries such as OpenCV 3 and PiCamera 1.13. In the main function LED pins are defined used for warning when a scratch is spotted. The special function *with* ensures that camera object is created and deleted after operation is done. It is important to release the camera in order not to cause collision. Camera attributes are assigned as mentioned in Sec. 4.4.2.1. At this point, camera module need some time (2 seconds) to settle some attributes.

Finally, a continuous capturing loop starts consist of main method (Fig. 4.9), time ticks (for performance measurements) and optional print of resulting image.

Here is the flow of the main processing algorithm (Fig. 4.9):

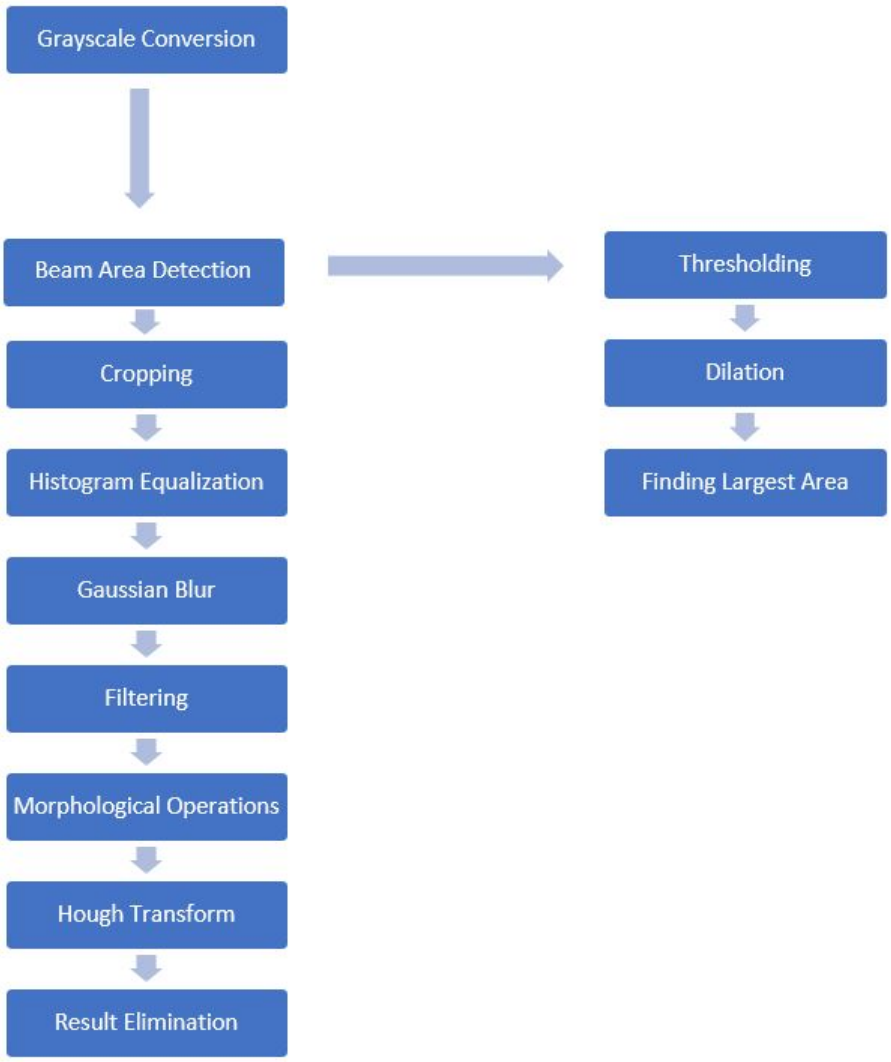


Figure 4.9: Main method of low-cost system algorithm

4.4.2 Implementation of Low-Cost System

In this section, an overview of the proposed low-cost system will be given. This system consist of a mechanic frame, a light source (more can be added if needed) and a *RaspberryPi 3* small computer that is upgraded with a *Camera Module v2.1*. Resulting system is expected to be successful with an acceptable hit rate, portable, inexpensive and fast enough to be considered as real-time.

Algorithm is written in Python 3.4 language using Picamera 1.13 Library and in OpenCV 3.4.1 framework. It takes some time to prepare the required working platform on Raspberry Pi for such a task, OpenCV and PiCamera libraries are really useful in the end, if successfully build in the device. Also, Python language was chosen over C++, because it is much easier to develop a research project like this while coding neatly and short. But it suffers from 30% speed drop compared to C++, so that changing language in the final product could be a future consideration 5.2.

4.4.2.1 Image Acquisition

Even though, there is built-in library of Unix terminal for camera control, a Python script is always easier for changing attributes and modeling capture methods on Raspberry Pi. For this purpose, a third-party library named *picamera* was used, and it was the only proper choice covering all the features Raspberry Pi Camera Module has. It is also more convenient to use a Python library, as the main algorithm runs in OpenCv Python. The picamera library is constantly growing and still matchless in terms of facilities.

The class picamera.PiCamera is the main class of image capture library. This library provides Raspberry Pi camera (both version 1 & 2) to be controlled by a Python interface. Camera is initialized with inserted parameters to the constructor function. Some of the camera properties can be altered after, even during real-time loops. On the other hand, some properties are capture operation related, such properties should be handled during capture function calls. All in all, camera object construction, parameter adjustments and capturing are 3 main steps.

Camera initialization requires some of the properties to be selected properly. Unless the user is willing to continue with default values; then, sensor mode, resolution, frame rate and clock mode must be indicated while construction phase. Setting those in the beginning is a time saver.

In this section, all necessary properties and capture functions will be discussed. All information is gathered from the official document page of PiCamera library [5]. An important note, camera must be released by close method (alternatively with *with*

structure of Python).

After setting up Python and picamera environment, camera module is attached properly as shown in Figure 4.10 and also camera is enabled by GUI or *raspi-config* command.

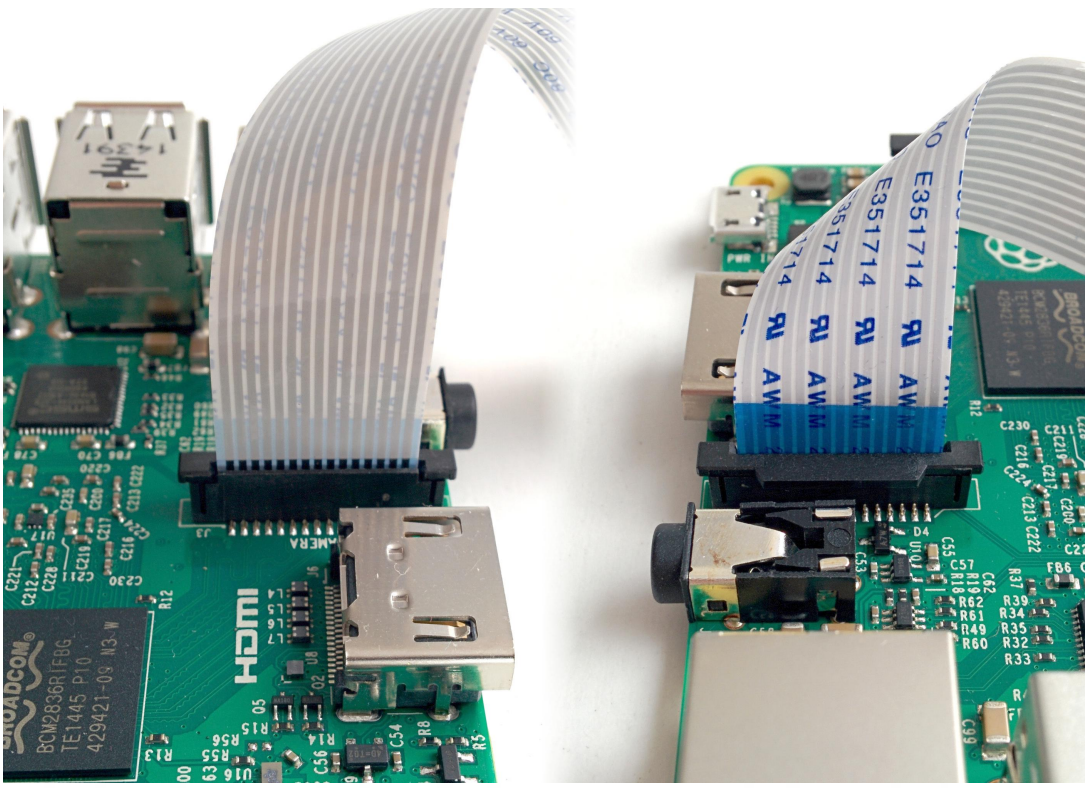


Figure 4.10: Proper connection to CSI port with correct direction

Camera Class Parameters

This part includes adjustable camera settings with explanations and some examples of resulting affects. Section below may perceived as unnecessary, but in fact a long time was dedicated understanding how those parameters effected the raw image we acquire. Parameters were kept changing as algorithms developed, all in all it almost took half of the time spent for experiments. The motivation was the fact that even the best line detection algorithm would fail, if images acquired were not appropriate.

Here below, important camera options of RaspberryPi Camera Module v2.1 (Sec. 3.3.1) are given. Background information was given in Section 3.1:

Sensor Mode is a special parameter of Raspberry Pi camera. This attribute indicates the mode of sensor output directly flowing to GPU. The Table 4.1 presents all possible sensor modes for V2.1 module.

Table 4.1: List of sensor modes for V2.1 Raspberry Pi camera [5]

#	Resolution	Aspect Ratio	Framerates	Video	Image	FoV	Binning
1	1920x1080	16:9	1/10 <= fps <= 30	x		Partial	None
2	3280x2464	4:3	1/10 <= fps <= 15	x	x	Full	None
3	3280x2464	4:3	1/10 <= fps <= 15	x	x	Full	None
4	1640x1232	4:3	1/10 <= fps <= 40	x		Full	2x2
5	1640x922	16:9	1/10 <= fps <= 40	x		Full	2x2
6	1280x720	16:9	40 < fps <= 90	x		Partial	2x2
7	640x480	4:3	40 < fps <= 90	x		Partial	2x2

In this Table 4.1, every mode has a sensor resolution and FPS determined. But still, user can enter a resolution with FPS manually, then a ISP chip would try to resize image to fit forced resolution within hardware limitations. Any of these sensor mode can produce video port frames, but only max resolution (3280x2464) cases are able produce an image port quality output. Depending on given sensor mode and resolution, ISP will try to interpolate or down-sampling. In addition, inserted FPS value will be force in possible ways, it was specified apart. Some sensor mode has impact on FoV, that may be an undesirable result.

In summary, we work on *sensor mode # 5*, because it gives full span in horizontal axis (crop in vertical is not important for our case) and quality is satisfying enough to observe linear details. But, resolution is forced to *640x480* in order to capture frames as quick as possible without losing required amount of detail quality. FPS is set to value of 40 to force the device to its limits.

AWB Mode stands for Auto-White Balance Mode. Adjusts two valued white balance gains of the camera. This can be done manually by setting this 'off' and inserting awb_ gains. In any case, AWB may be affected by *still_stats* and/or *drc_strength* parameters.

This parameter was chosen as *flourescent*, because of the color of the light we work under. AWB mode does not cause significant differences, though. Yet, it was not enough controllable, if AWB was in its default mode *auto*. Please, refer to Section 3.1.2 for further information.

Brightness is a parameter that adjusts total white value of the image. Increase in brightness will bring about a luminous view, whites to get whiter and dark details to get visible. In our application, this has no important effect on scratch detection; however, with brightness value of +50 (default 0 in [-100,100]) was selected to compensate dark image caused by low shutter speed. The example in Figure 4.11 shows effect of brightness, it was 0 (default value), image would be completely black. But in the figure, light source is vaguely seen.

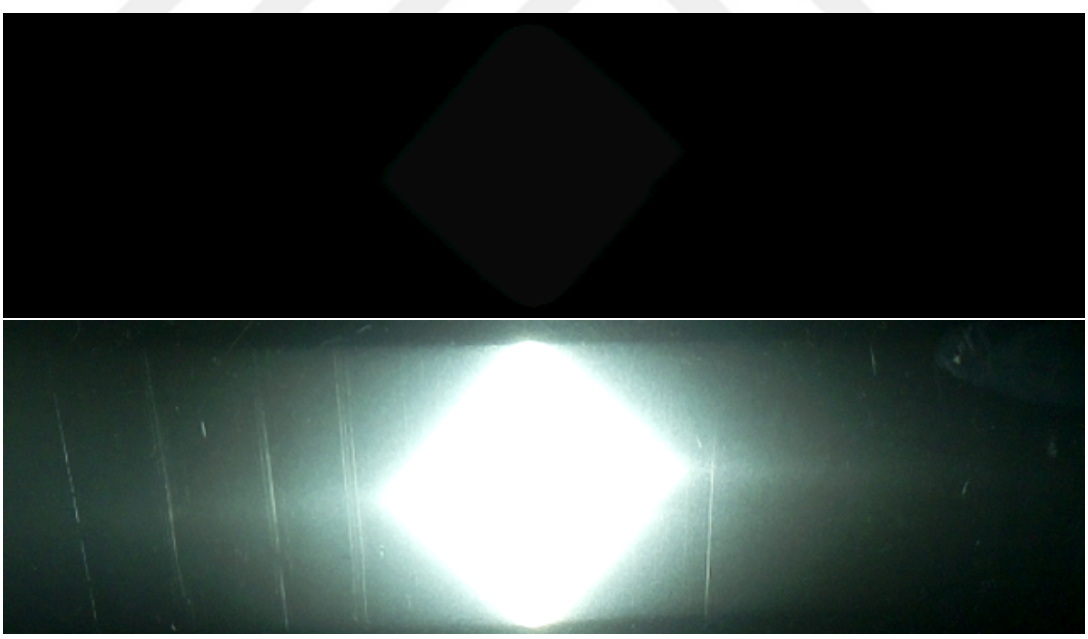


Figure 4.11: Effect of brightness (values = 5, 50)

Contrast is another essential attribute that can be encountered in every image related application. It can take values in [-100,100] interval, 0 being the default. We preferred to keep it that way as higher contrast give better details with much more noise, while lower contrast has no meaning in our application. Please refer to related Section 3.1.3. In the Figure 4.12, there given comparison of default contrast and contrast value of 50 images. Details get more explicit, but image becomes noisy and bright visible area gets smaller.

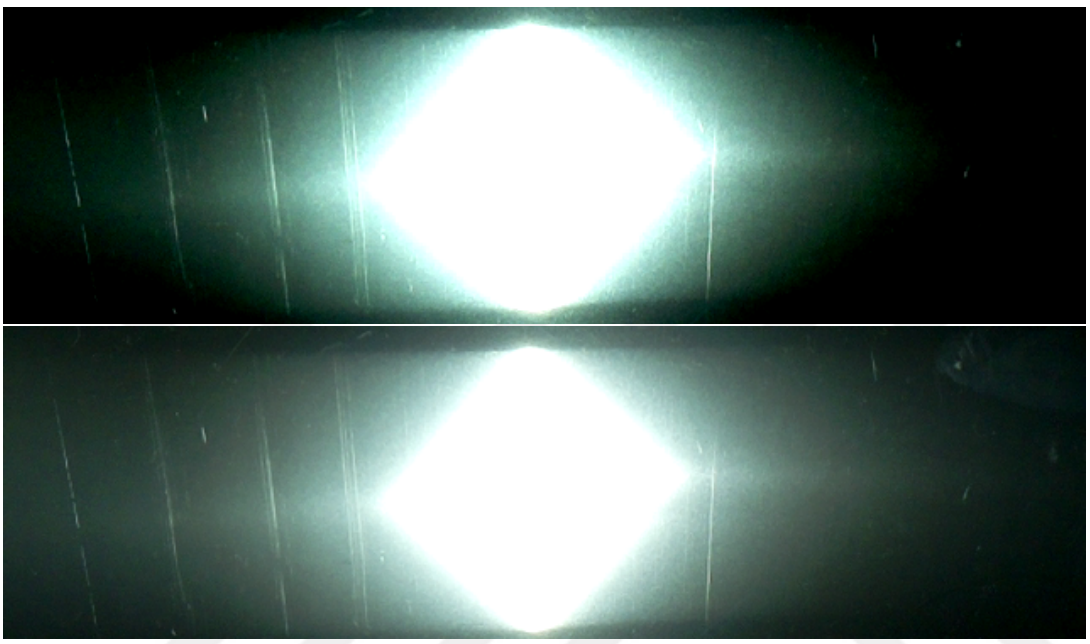


Figure 4.12: Effect of contrast (values = 50, 0)

DRC Strength stands for dynamic range control strength. When set, this parameter causes device to run algorithm of dynamic range, which strives to brighten dark area, conversely make shiny parts less luminous. As a result, all the detail in extreme intensity parts get visible. As for our application, a high DRC would make sense, so that it would adjusted to *high* setting. Although, the direct impact was not visible. Please refer to related Section 3.1.6. As a note, we implement an adaptive histogram equalization method which aims the same effect as high dynamic range. Alternating DRC results in overriding AWB gains, too. But there is no clear information on how much it affects on the gains.

Exposure Mode , with the co-operation of ISO parameter, controls digital and analog gains of the camera regarding how responsive sensors will be to the light. As expected exposure mode has impacts on ISO and shutter speed. As usual, the options for this mode have names of common scenery such as night, sports, fireworks. Not experiencing a significant effect (as we manually force shutter speed and ISO in parallel), we have set *backlight* to exposure mode. As in AWB case, we wanted this to be a fixed value other than *auto* (default). Please refer to related Section 3.1.5 for further information.

Exposure Compensation is explained in Section 3.1.5, too. The changes in this parameter was not observable, so we set +25 to get brighter images theoretically.

Image Denoise is another special ISP chip parameter which takes boolean values. Setting this to *True* gives a small amount of denoise in acquired frame.

Image Effect is yet another parameter of Raspberry Pi camera. This is in fact intended for artistic application like solarization, negative images, watercolor and pastel effects. Several modes has their own secondary parameters. For this, we again decided on *denoise*, which has no significant change in practice, yet a clever choice in theory.

Meter Mode is a tool for evaluation of correct exposure. The alternative in this mode selects a ROI on the image for exposure calculations, then to get optimal exposure, shutter speed and aperture are adjusted automatically. As we manually set those parameters, meter mode has little impact in the images we have. But, still as shown in Figure 4.13, *matrix* (divides plane into grids), *backlit* (30% central region for exposure measurement) and *average* (20% central region for exposure measurement) are proper choices. The option *spot* concentrates on a small region in the center (10%) and gets easily deceived, because normally there is bright light source is located in the middle of an image.

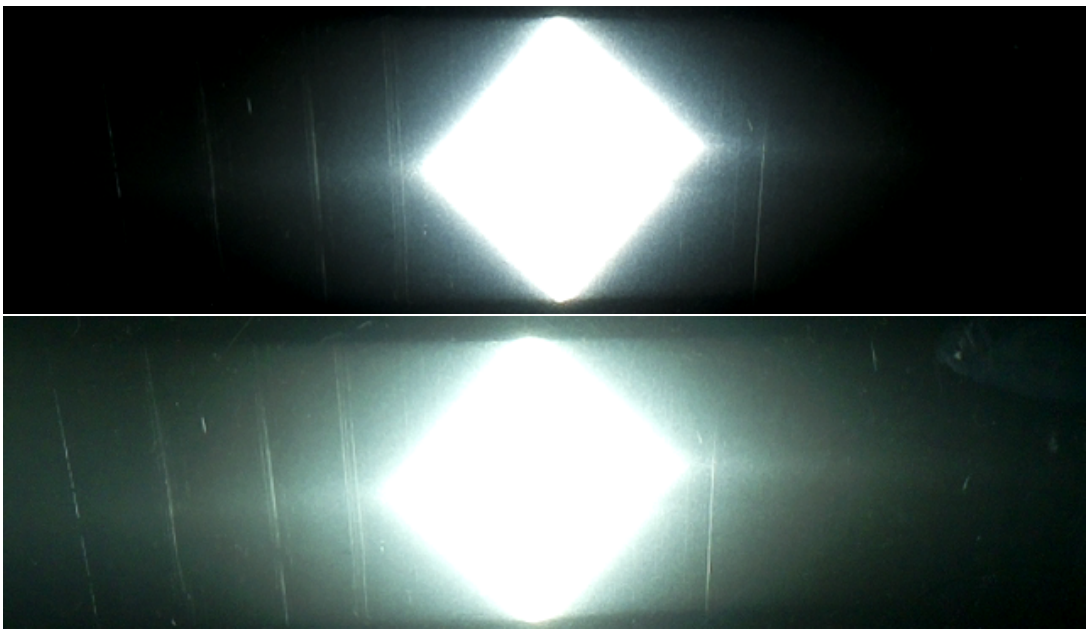
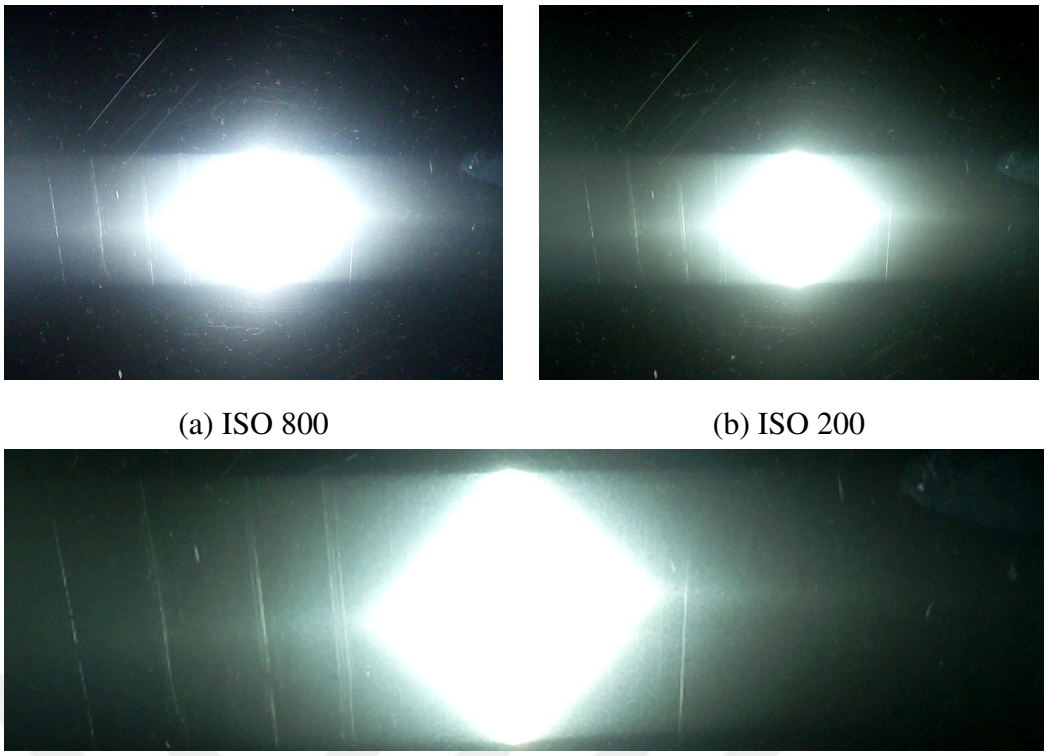


Figure 4.13: Effect of metering mode (values = spot, backlit)

ISO is an important parameter in such application where luminance plays a big role. In the favor of theory, as ISO goes high sensors get more sensible to light and details are observable a lot. But, as a downside, noise is amplified. In contrast, a low ISO results in low noise which is more helpful in our illumination level (Fig. 4.14). Background information concerning ISO is in Section 3.1.1.



(a) ISO 800 (b) ISO 200

(c) ISO 100

Figure 4.14: Effect of ISO (values = 800, 200 and 100)

Saturation is a parameter determines how colors are intensifies in an image. This has no value in our gray scale application and left as 0 (default).

Sharpness is also an essential parameter of visual devices. It was explained briefly in Section 3.1.7. For scratch detection purposes, even though SNR degrades, we needed to have maximum sharpness of 100 to obtain sharp edges around linear objects (Fig. 4.15).

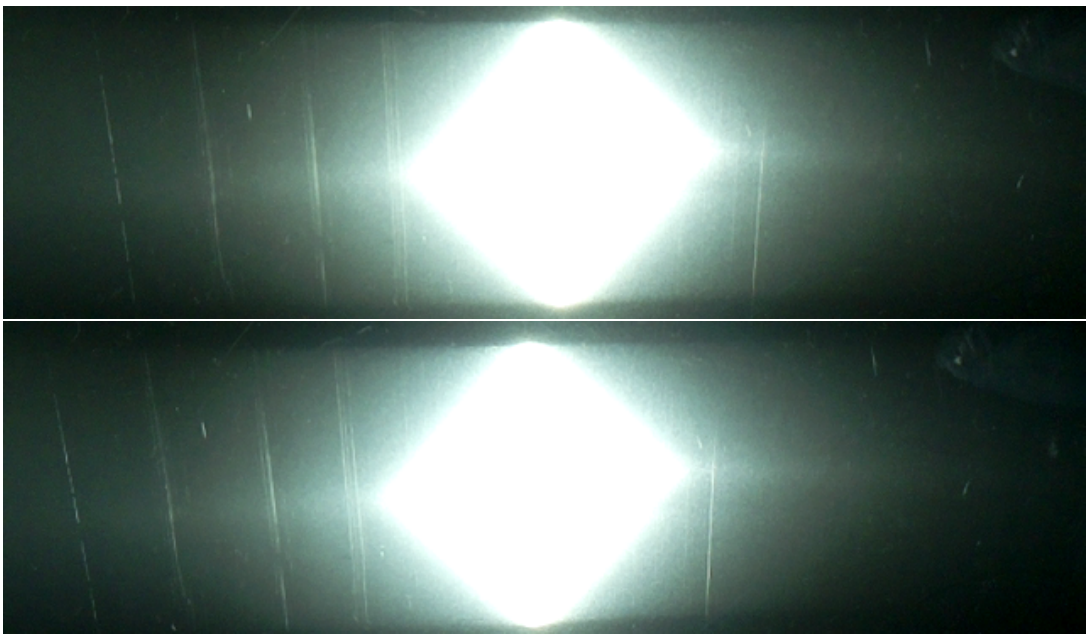


Figure 4.15: Effect of sharpness (values = 0, 100)

Shutter Speed of Raspberry Pi camera is not a totally accurate terminology. As stated before (Sec. 3.3.1), there is no actual shutter closing in this device, but there is *rolling shutter*. This technique is well explained in 3.1.4. In practice, faster shutter speed requires more illumination, but creates less motion blur. The values are in microseconds, zero value depicting auto shutter speed mode.

We decided to go down as possible as we could in microseconds. Because, high shutter speed eliminated motion blur completely. Before, due to blurring it was impossible to catch scratches from metal sheet moving on the line. After some experimenting, we fixed this parameter to 2500 microseconds. Going lower causing a lot darker images and failed in low light conditions (Fig. 4.16).

Figure 4.16: Effect of shutter speed on moving image (values = auto(slow), 1000(fastest) and 2500)

Still Stats is an attribute concerning image port application, while in our case it is video port. To explain it anyway, it will determine if current frame or previous frame is used to calculate scene statistics, e.g. exposure. When *True*, statistics are calculated from the captured image and time between star up and capture is reduced. As a disadvantage, capture time increases. This dummy parameter is set to *False*.

Video Denoise is similar to image denoise parameter, but in this case video port frames are denoised by ISP chipset. Naturally, this attribute is switched to *True*.

Video Stabilization is a special Raspberry Pi camera function that aims to reduce horizontal and vertical motion blur. This is must choice for our case and set as *True*.

Table 4.2: Camera attribute values set manually (for low-cost method and data set)

Attribute	**Value**
Resolution	640×480
Frame Rate	$40\ FPS$
Sensor Mode	#5
AWB	*fluorescent*
Brightness	$50 \in [-100, 100]$
Contrast	$0 \in [-100, 100]$
Exposure Compensation	$25 \in [-25, 25]$
Exposure Mode	*backlight*
Image Denoise	True
Image Effect	*denoise*
Meter Mode	*backlit*
ISO	100
DRC	*high*
Saturation	$0 \in [-100, 100]$
Sharpness	$100 \in [-100, 100]$
Shutter Speed	$2500\ msec$
Still Stats	False
Video Denoise	True
Video Stabilization	True

Capture Methods

These functions, basically, capture an image frame or several frames, either to be stored in a dynamic array or to be written to a proper file. This differentiation is indicated by the very first argument, namely *output* argument indicated in capture

function call. If it is a string or a variable with *write* method, then captured frames are written to a file. Otherwise, buffer protocol is initiated and image is stored dynamically. Physically, camera constantly captures images for gain and exposure control, but when command is send for capturing, sensed image array would not be discarded. It will be transferred to ISP unit, instead.

In this context, it should be pointed out that there is a software layer of PiCamera library called MMAL. This layer makes an interface with GPU firmware and controls *ports* of the camera module. These ports are, namely, *still port*, *preview port* and *video port*. Preview port, as the name suggests, is used for previewing and not qualified enough for processing jobs. The still port, on the other hand, serves the best quality. Please remember that, only best two resolution sensor modes can use still port, i.e. image mode (see Table 4.1). The problem with this port is that, the captured image, to be transferred to ISP unit is in 3280×2646 resolution. This cannot be downgraded to low resolution that we want to work with. Alternatively, another software resizing should be added to the algorithm. All in all, whole calculation time increases unnecessarily, even though we have better quality images due to strong noise reduction of still port.

The port we used is *video port*. This port is activated by choosing *use_video_port* parameter *True*. This port obviously causes FPS to be greater compared to that of still port. But, noise reduction is somewhat lesser, and frames captured are tend to be grainy and noise compared to still port images. This port allow us to work with low resolutions faster. In low-cost system, the real-time FPS was a more delicate issue, then we decided to go with this port, as total time (calculation and capture time together) is 3 times as it was for still port.

Finally, the capture function itself was to be selected among three choices: *capture*, *capture_continuous* and *capture_sequence*. The first is suitable for snapshots, meaning single still image acquisition, while the third is for a number of images to be captured consecutively. Function we use is the second choice, that constantly captures images in a non-blocking way, so that there is no time spend for camera wake-up.

Capture functions requires some input arguments. We save frames into special PiCamera buffers (*PiYUVArray* in this case). Video port is activated with related pa-

rameter mentioned above.

The arrays, transferred to software buffer, consist of *raw* image data, in other words, there is no compression due to image format change. Moreover, the codec used is YUV, in this case. The default is three color channel BGR mode, though. The reason is timing , simple time measurements are given in Table 4.3. Those timings include only image capturing and image demonstration on screen. Still, there is an extra step in BGR case, which is conversion to gray scale that is the format we work with. For YUV case, the first channel has already single channel luminance information. In short, YUV gives faster result and seems more reasonable.

Table 4.3: Time measurements for previewing BGR and YUV raw images

	YUV	**BGR**
Image Show Time (on screen)	0.009 sec	0.007 sec
Total Time	0.09 sec	0.10 sec

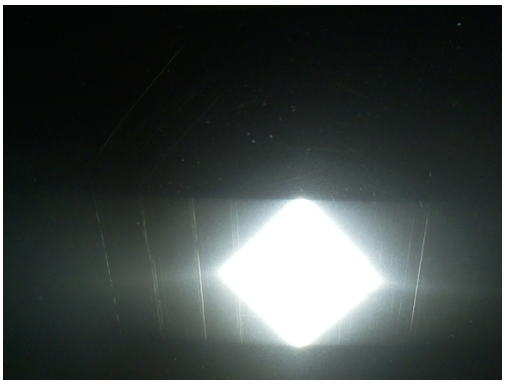
Please note that, for data set acquisition, the images were needed to be saved into hard drive in 3-channel bitmap file format.

4.4.2.2 Algorithm and Hardware Results

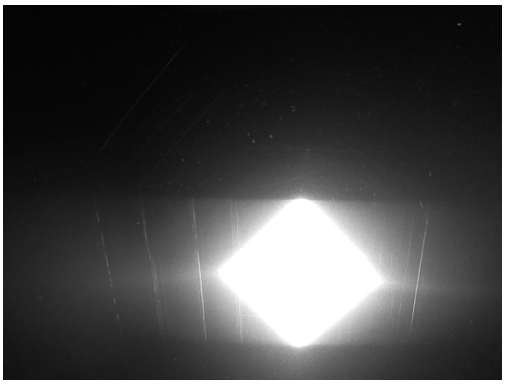
In this part, implementation details of the low-cost system directed to problem (Chp. 1) will be presented on Raspberry Pi 3 hardware (Sec. 3.2.1) supported with method provided in Sec. 4.4.1.

It should be noted that using *raspi-config* command total memory dedicated for graphical interface was doubled and made 256MB.

After capturing a frame successfully, we need to convert it into gray scale immediately, as color information would not be used. For YUV format capturing, this is merely picking the first channel which has luminance data (Fig. 4.17a-b).



(a) BGR Raw Image



(b) Only Y Channel

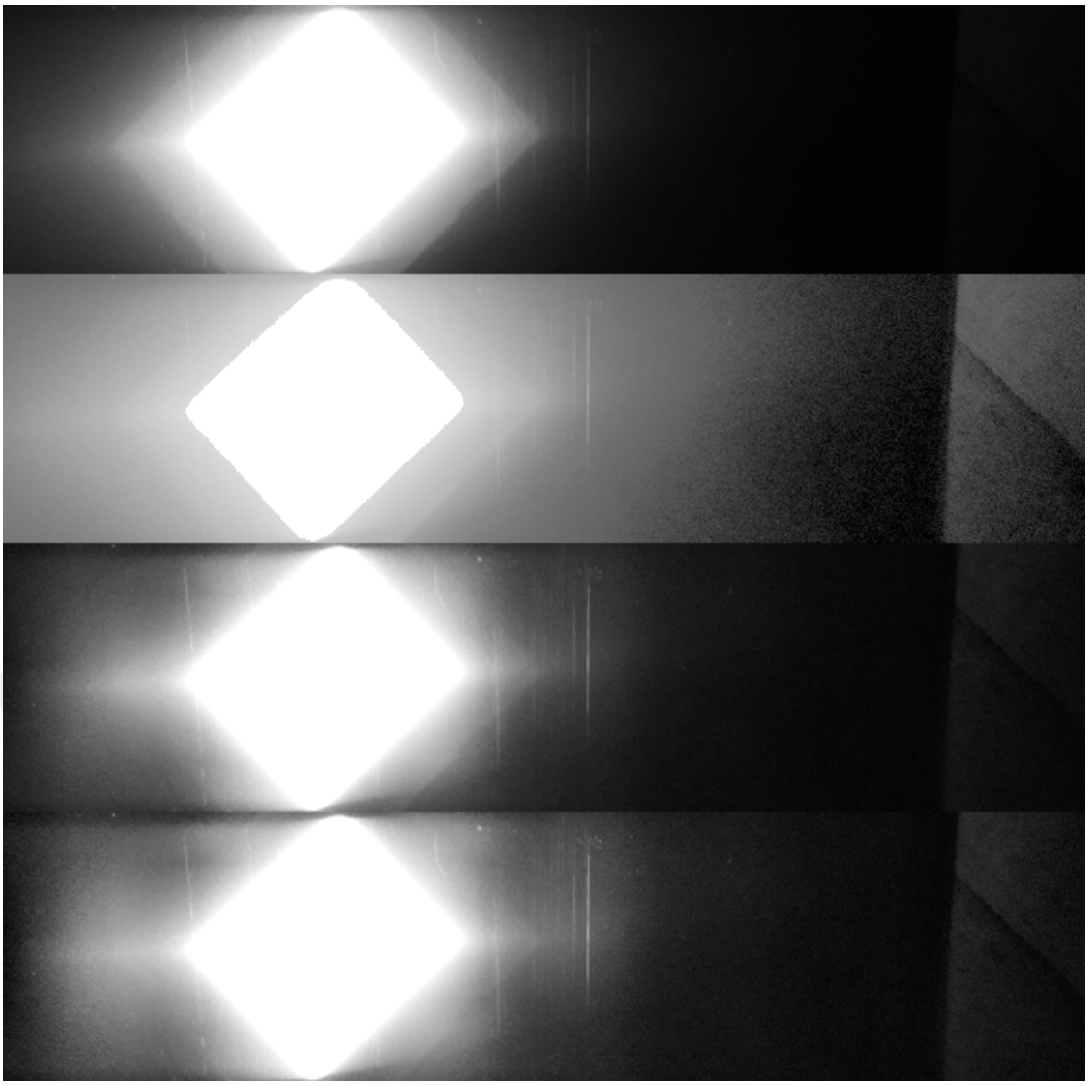
(c) Thresholded and Dilated Image

Figure 4.17: Raw image capture and detection of beam area

Then, we skip to *beam area detection*. As mentioned before, we use the phenomenon that scratches appear in the beam area clearly. To spot this area, we have to locate light source in the image first. The light source has an evident feature, being the largest bright object. To get the help of that, whole image is thresholded with lower limit of 215 (in [0, 255]). This number is found experimentally. The perimeter of

light source is covered with black paper, so the closed system here lets us to use such a fixed number. Alternatively, an Otsu Threshold (App. A.1.2) could be used here with a fixed coefficient. The mask acquired is dilated slightly to close small holes (Fig. 4.17c). In the final step, the contour with the largest area is found owing the an OpenCV built-in function. This function returns coordinates of the bounding box around the largest area, which is the light source, if camera directly illuminated by it. Then, the gray image is cropped according to that bounding box, but original dimension in horizontal axis is preserved.

The very next steps are pre-processing steps in order to transmit an image to Hough Transform, consisting vague linear features. This starts with equalizing the histogram of the cropped gray frame. Please note that, this image has only one channel of data, which is intensity. Histogram equalization (see App. A.1.4, conventionally tries to match histogram graph of an image to a uniform one. This means that whole image is normalized in this case. Additionally, by experiments and theory, an adaptive histogram equalization is a better choice. In this manner, *CLAHE* method is preferred, that slides a window around the image and treat every part of image differently. *Adaptive* word suggest that sliding window is uniquely calculates histogram everywhere. It is also *contrast limited*, so user can define a contrast limit in order not to amplify noise in small fragments of image (App. A.1.4. In short, there are two parameters of CLAHE, tile size and contrast limit (or *clip limit*).



(a) No Histeq (b) Uniform Histeq (c) CLAHE(2, 8) (d) CLAHE(3, 8)

Figure 4.18: Difference in Histogram Equalization Methods

In Figure 4.18, an original cropped image is given with a uniform conventional histogram equalization under. The last two are CLAHE images with alternative contrast limits. The suggested global parameters for CLAHE are 8×8 tile size and 3 to 4 contrast limit [92]. Figure 4.18c shows CLAHE with 8 pixels tile edge and 2 clip limit. Figure 4.18d shows CLAHE again with 8 pixels tile edge, but 3.5 clip limit. As the demonstration suggests, adaptive equalization gives better results, and as clip limit goes up details get stronger but noise is amplified in the same proportion. We preferred 3 contrast limit in our application, but in variations of the code this param-

eter can be changed to 2 or 4. On the other hand, bigger tiles give rise to significant increase in calculation time and undesirable results. In the end, we continued with 8×8 sliding windows.

$$\begin{bmatrix} 0.0017 & 0.0026 & 0.0017 \\ 0.0169 & 0.0268 & 0.0169 \\ 0.0675 & 0.1069 & 0.0675 \\ 0.1069 & 0.1693 & 0.1069 \\ 0.0675 & 0.1069 & 0.0675 \\ 0.0169 & 0.0268 & 0.0169 \\ 0.0017 & 0.0026 & 0.0017 \end{bmatrix} \tag{4.5}$$

After histogram adjustments, filtering is to be applied, but *Edge Detectors* are highly unprotected against noise in the images. So, a blurring technique has to be applied at this point, which would be *Gaussian Blurring* in our case. The lines we seek for are vertical lines, so a Gaussian kernel in a vertical rectangle form is applied with size 7×3 (Matrix 4.5). As a result, in vertical axis, noise are reduced more, but vertical details are not critically effected. A bigger Gaussian may give better results in terms of SNR, but it can also remove thin horizontal details of the lines and this operation would be much slower. This parameter may still be changed in variations. The sigma parameter of Gaussian function was calculated automatically according to the kernel size, which is not a significant argument. Please, refer to Appendix A.1.5 for further details. Necessity of blurring is clearly shown in Figure 4.19.

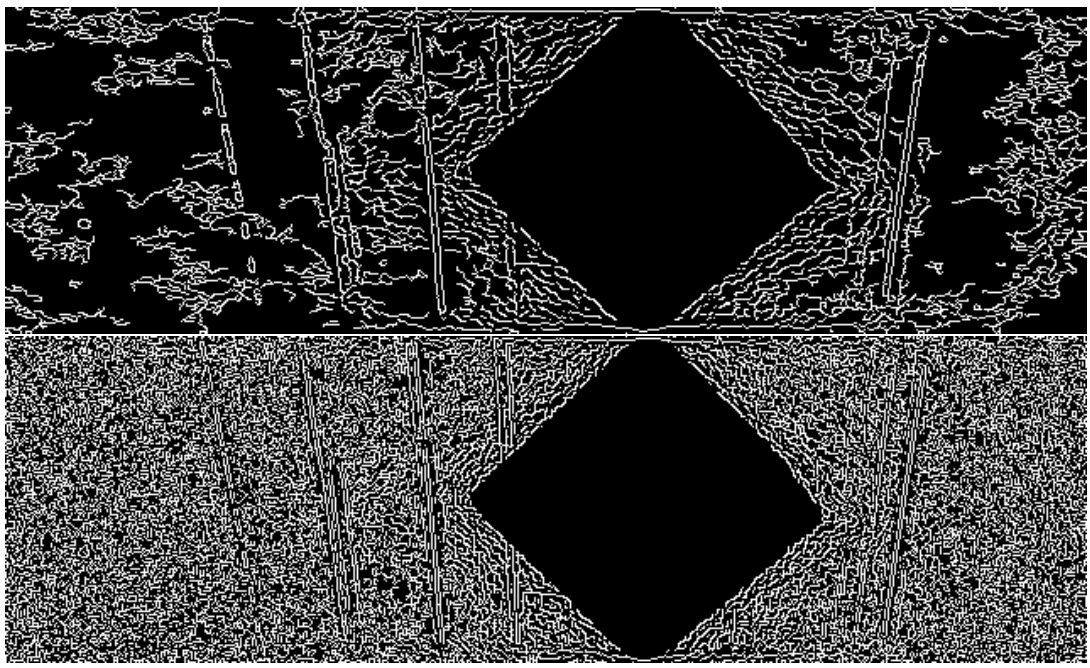


Figure 4.19: Canny Edge Filter results with and without Gaussian blur (7×3 $kernel$)

To that blurred image, now an edge detector may be applied. The edge filter chosen was *Canny Edge Detector* ,because it applies a low-pass filter to eliminate speckle noise and is able to catch weak edges connected to stronger ones (App. A.1.1). A resulting image of Canny filter is already shown in the Figure 4.19a, with the help of parameters $1 and 35$, thresholds of weak and strong edges, consecutively. Note that, even the strong threshold indicated is a small one. It is due the fact that scratches are sometimes hard to spot by edge filters, caused by reflectivity. In addition, Canny Filter kernel is 3×3 in our case. No small detail should be lost. Variation may include a Canny filter with different parameters, a Sobel filter for vertical details instead or both filters in the same implementation.

As we chose rather low Canny thresholds, it was needed to filter noise related edges. At this step morphological operations come in action. This part, in fact, was completely experimental. The main intention was to eliminate noise related details that has no vertical thickness, but a few pixel long horizontal lines. The resulting image (Fig. 4.20) shows clearly, only vertical components are reserved after applying an *image opening* algorithm with a 3×1 kernel. This part is always open to improvements.

Image opening, closing, dilation and erosion may be used, combined and added up under different environmental conditions with varying kernels both in shape and size (App. A.1.3).

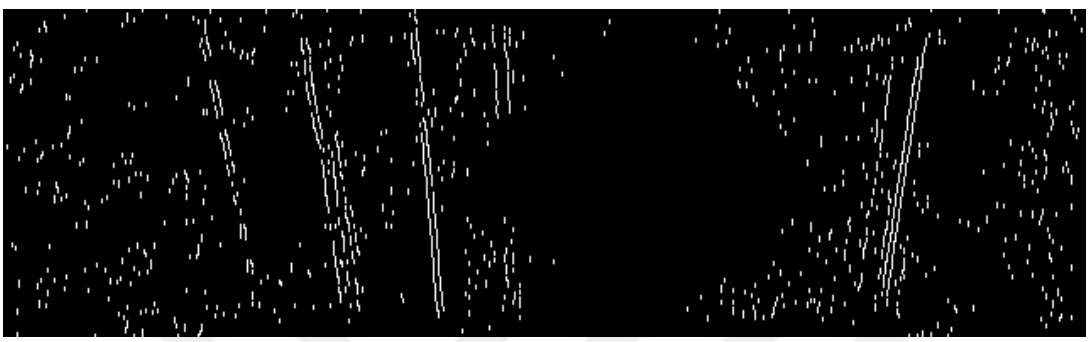


Figure 4.20: Applying image opening with 3×1 kernel

In the final image so far, the vertical linear features are fairly visible. Now, it is time to apply the main method *Hough Transform* to locate potential line objects. Applying conventional Hough Transform (Sec. A.2), all the linear features are obtained. Here, rho resolution 1 pixel, while $theta$ resolution is $\frac{\pi}{180}$. From Hough space matrix, line nominees over threshold of $\frac{h}{4}$ are picked, h being the height of the cropped image. In other words lines with the length of one fourth of the current image are considered. As a final filter, line objects with non-vertical angles are eliminated. Only lines with upright angles within a 40 degrees cluster remains to prevent false results in the vertical axis. The Figure 4.21 shows the linear results drawn on the image as an overlay.

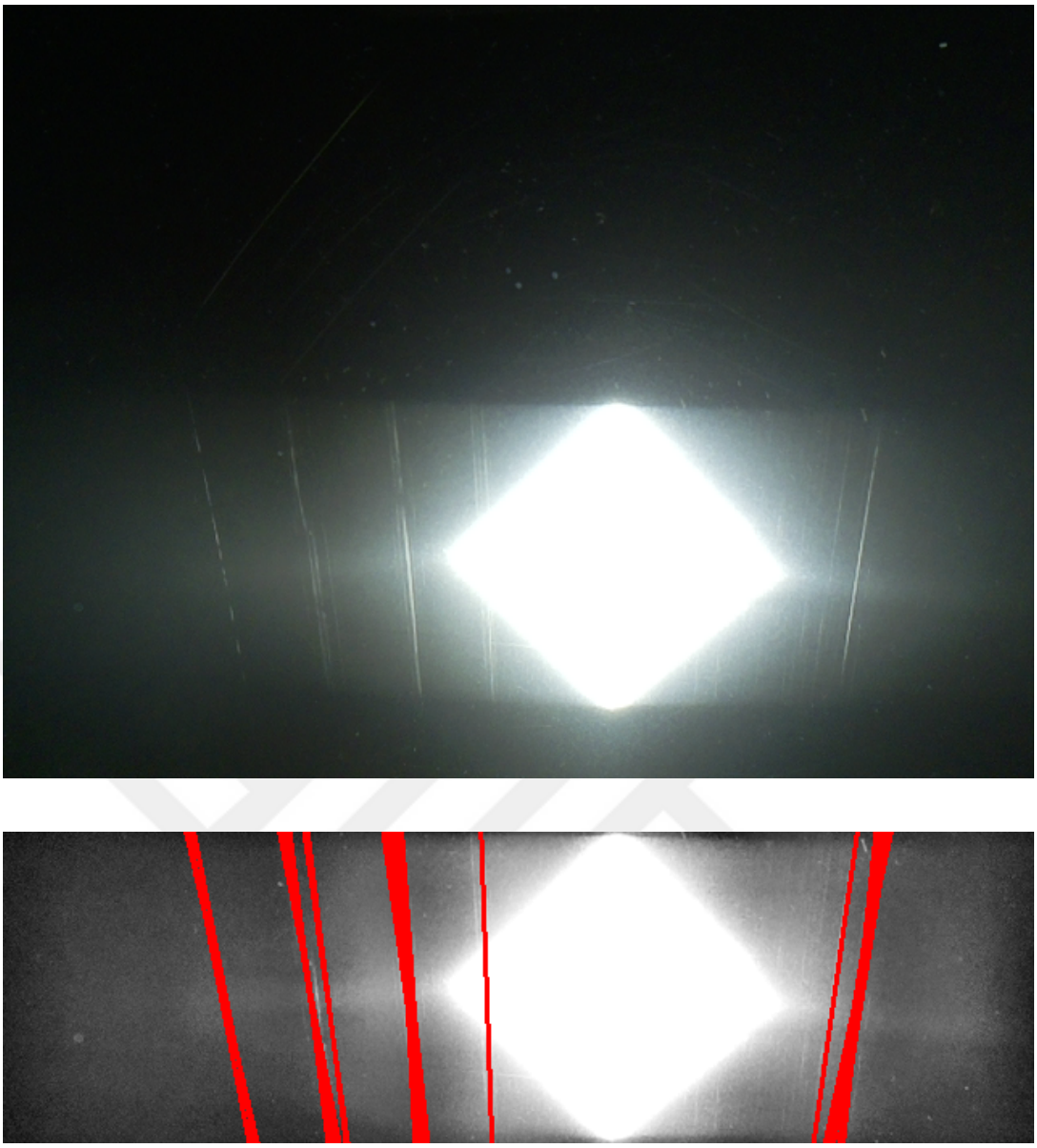
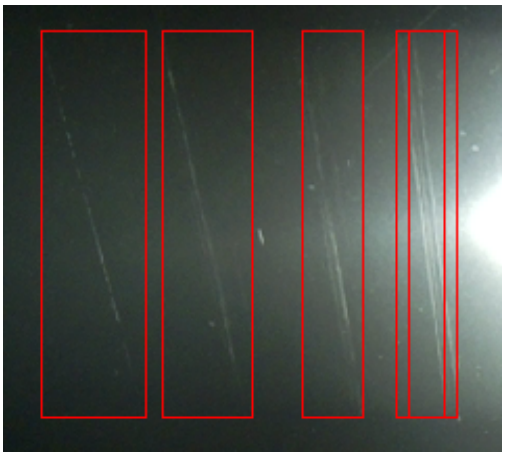
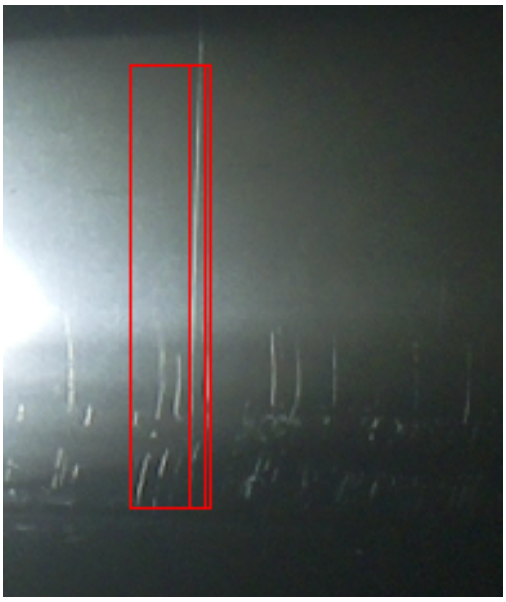


Figure 4.21: A resulting image with its input image

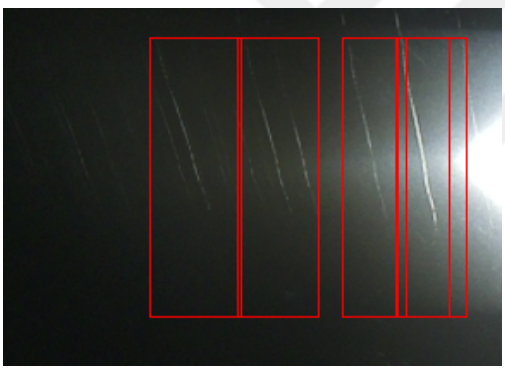
After this point forward, still, some more improvements may be realized in the needs of customer or application. For example, not being happy with the visualization of the results, we instead drew rectangles. Additionally, while doing so, the lines with center points close to each other, are merged. In the resulting configuration, there is not many rectangles overlaying each other in a confusing way. Here below in Figures 4.22, there are some examples of input metal sheets and results of low-cost system:



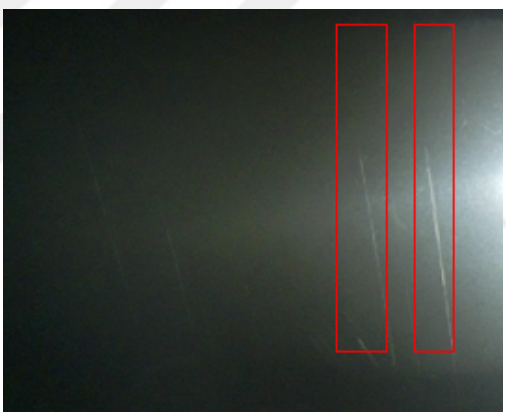
(a) Resulting Image 1



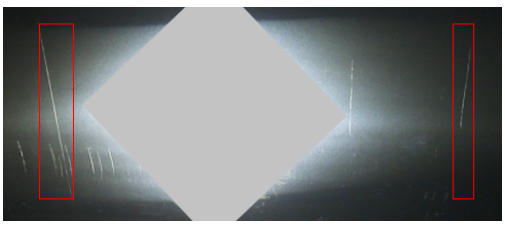
(b) Resulting Image 2



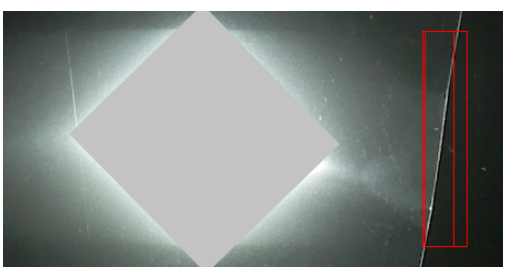
(c) Resulting Image 3



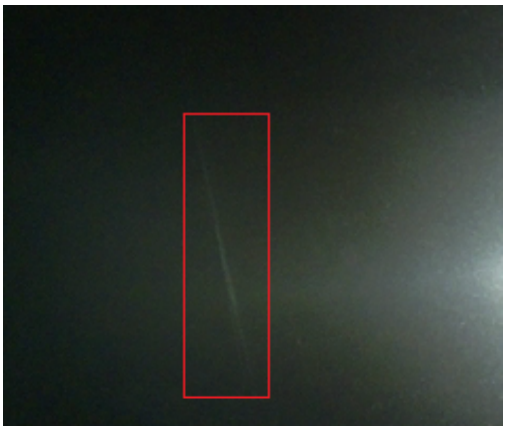
(d) Resulting Image 4



(e) Resulting Image 5



(f) Resulting Image 6



(g) Resulting Image 7

Figure 4.22: Low-cost method hardware implementation final results of the main method

Examining the results in Figures 4.22, intuitively it can be said that low-cost solution on Raspberry Pi 3 with its camera module seems to accomplish the task sufficiently, considering its cheap price. To be more precise, Fig. 4.22a is the one example of which earlier steps are shown in this section. This case is chosen as the main example, because it involves some different types of scratches together. As clearly observed in Fig. 4.21a, there is a several times broken line (1st from right) and a line overlapping light source (4th from right, in the middle). Those two types are the most difficult to locate, in fact. The other three scratches are detected easily. Please note that, insignificantly seen scratches are also found in the resulting examples. The example Fig. 4.22f involves a scratch like line on the right hand side, which is actually the border of the metal sample. Yet, that edge appeared white and considered as a linear object. The whole method is discussed in Sec. 4.4.2.4 in terms of performance and timing.

4.4.2.3 Temporal Aliasing Problem

The problem originates from video noise. Probably, shot noise [93] or film grain [94] is main responsible for false results. Those noise types majorly specified by sensor quality, furthermore, we acquire high sharpness, high shutter speed (less brightness)

and low resolution frames, which causes noisy results. Even though it is adaptive, histogram equalization always amplifies certain amount of noise. I also suspect from *rolling shutter* technology of the camera module (Sec. 3.3.1).

To define the problem, when there is a still metal sheet on the line, the still video may produce frames so differ that distorts the result. Consequently, it gets hard to adjust parameters for a consistent solution, because noise fluctuates a lot. An example is shown in Figure 4.23. The resulting image goes between given outcomes, without a pattern. The broken line overlapping light source is detected without succession.

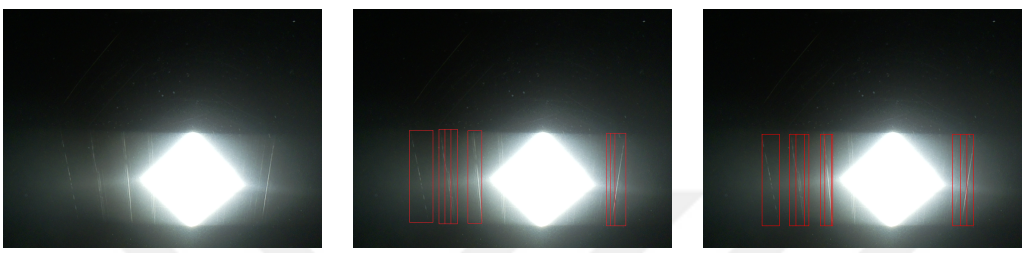


Figure 4.23: Temporal Aliasing Problem

The first and definite solution would be changing the camera, but we aim to solve it low-cost as a rule of thumb. Still, a better quality sensor should solve the root problem. In terms of software, it can be solved with a trick. This would be recognizing a scratch, only if we encounter the same scratch in the same location in 3 frames in 5 consecutive frames. Those number can be changed by experiment, but the main logic is create a temporal filter in a way.

4.4.2.4 Conclusion

This section is dedicated to final words on low-cost method results. All in all, the system costs around $ 60, which is a really inexpensive solution for such task. Although, it is the only way to test a hardware implementation to visually check if it can find scratches in desirable number of successful cases. At this point, the requirements of customer is involved and an inspection expert may be needed. Still, according to our observations, the system is correct for around 90% of the cases.

During this work, we produced in total of three variations of the method. The first one is the main approach, which is explained and illustrated with examples in this whole section. The second one still works with Canny filtering, but programmed by different parameters. The last one has Sobel filtering which more primitive compared to its counterpart. Sobel filters only emphasizes features in one direction, which may seen enough, however without low pass filtering and weak edge detection function. Yet, necessary details for all variations are given in the Table 4.4. The main method is optimum regarding noise and true positive rate. The second, is more stable but sometimes fails to find broken lines. The third is fairly aggressive with high true positive and false positive results.

Table 4.4: Implemented parameters of low-cost system algorithm variations

	Main Method	**Variation 1**	**Variation 2**
Beam Area Masking Thr.	215	215	215
Histogram Eq. Method	CLAHE	Uniform	CLAHE
CLAHE (cont. limit, tile)	3, 8×8	-	3, 8×8
Gaussian Blur Kernel	7×3	5×3	5×5
Edge Filter Method	Canny	Canny	Sobel
Filter Parameters	1-35, 3×3	1-35, 3×3	2nd derivative, 3×3
Morph. Operations	Opening	Opening and Closing	Opening
Morph. Kernels	3×1	3×1, 1×3	3×1
Hough Resolutions	1 px, $\frac{\pi}{180} rad$	1 px, $\frac{\pi}{180} rad$	1 px, $\frac{\pi}{180} rad$
Hough Thresh.	$\frac{h}{4}$	$\frac{h}{3}$	$\frac{h}{4}$
Angular Filtering	$90° \pm 20°$	$90° \pm 20°$	$90° \pm 20°$
Notes	Success rate 90%, optimal perf.	Less broken line perf., better noise perf.	Worst noise perf., very aggressive

Simple idea but complexity of n parameters and m quantization bins is in the order of m^n. But, still this is a really cheap, portable, simple and less hardware burdening

system. In Table 4.5, given timing performance on real hardware. Please check pseudo-code (Sec. 4.4.2.2) for time tick locations in a loop. Note that tick 4 comes first, this is the breakpoint for total loop time. It is located such, due to Python implementation of camera capture is in *for* statement.

Table 4.5: Average hardware real-time measurements of low-cost system [sec]

	Main Method	**Variation 1**	**Variation 2**
Capture Time $(T_4 - T_3)$	0.089	0.089	0.089
Calculation Time $(T_2 - T_1)$	0.071	0.070	0.060
Show Time $(T_3 - T_2)$	0.018	0.011	0.024
Total Time $(T_4 - T_1)$	0.178	0.170	0.173

In summary, this method is very efficient in terms of price-performance considerations. The real-time implementation gives around 5-6 FPS (Tab. 4.5) with sufficient success rate of around 90%. But, it is far from robust, in many times, fine tuning of parameters would be needed depending on the material, scratch types and illumination of the environment. The major problem, namely temporal changes in the images, is explained in Sec. 4.4.2.3. Software based solution is also presented; moreover, a high quality camera would always sort out this issue. Some more future work idea can be found in the Section 5.2.

4.4.3 Simulation Results for Low-Cost System Algorithm

In this section, we will present Hough Transform based algorithm performance regarding timing and detection. As expected, this basic method was not very successful on finding weak scratches, but had no problem on stronger ones. Weak lines are either broken or has intensity values lower than 200 (out of 255) in a given image (Fig. 4.24). On the other hand, strong lines have high intensity values in the input image with length of at least half of the input image height. As the customer needs are to find those long and strong lines, in the final system, it will not poses a problem. All

in all, sensitivity values are much lower than second method's (See Sec. 4.5.3.2).

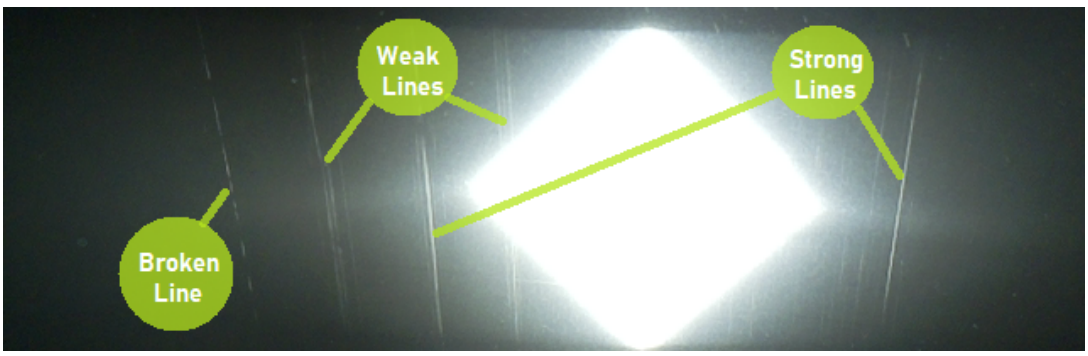


Figure 4.24: Demonstration of weak and strong lines

For this modeling task, MATLAB environment was chosen, due to the fact that it is very flexible for modeling, parameter adjustments and observing results. The modeling platform of low-cost system design is *MATLAB v2016a - Student Version* with *Image Processing Toolbox v9.4* on a *Windows v10 - Home* operating system.

4.4.3.1 Timing Results

This results, on the contrary of mid-end system, are expected to get better in hardware implementation. Because, Python implementation with OpenCV is much faster than MATLAB, but as stated before, MATLAB is the suitable platform for modeling. However, results show similar trends in terms of timing. In Table 4.6, the total loop time is almost the same with the ones in Table 4.5; although, they are implemented on completely different systems. The aim here is not to calculate real-world FPS, but to check if the final system is predicted to work in reasonable time. Still, time measurements in MATLAB model implies around 5 FPS.

The timing ticks are located slightly alike with Raspberry Pi implementation (Sec. 4.4.2.2).

Table 4.6: Average MATLAB modeling measurements of low-cost system [sec]

	Main Method	**Variation 1**	**Variation 2**
Capture Time $(T_4 - T_3)$	0.049	0.050	0.050
Calculation Time $(T_2 - T_1)$	0.028	0.030	0.026
Show Time $(T_3 - T_2)$	0.095	0.095	0.094
Total Time $(T_4 - T_1)$	0.173	0.175	0.170

In contrast with hardware implementation timings (Tab. 4.5), show time has taken a lot of time in average. But, in modeling case capture time is rather short, as it is not an actual image capture, but only acquiring it by reading a file.

Please note that, main method and its variations are defined in Table 4.4.

4.4.3.2 Detection Results

In this section, we will make use of semantic metrics and some visual examples.

As expected, Hough transform based pure image processing model, introduced in Sec. 4.4.2, has failed to achieve a high percentage in semantics. It is due to low resolution of scratches and their scale is rather small compared to whole picture. This is why we call this data set as *Hard Set*. As all the variations gave similar results, in average, we present only main methods results in the following way:

Sensitiviy 18.48%

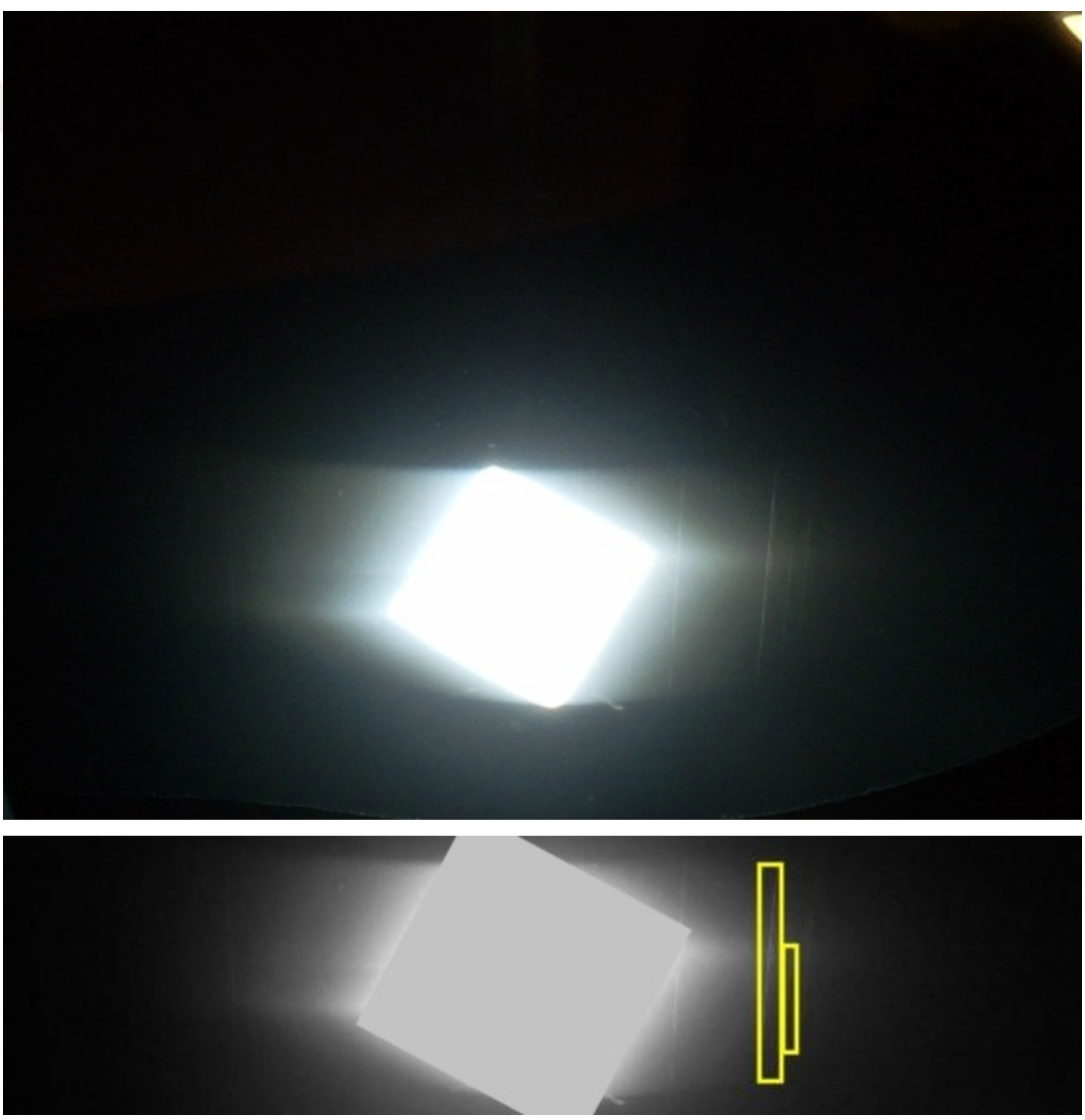
Specificity 98.40%

Accuracy 98.27%

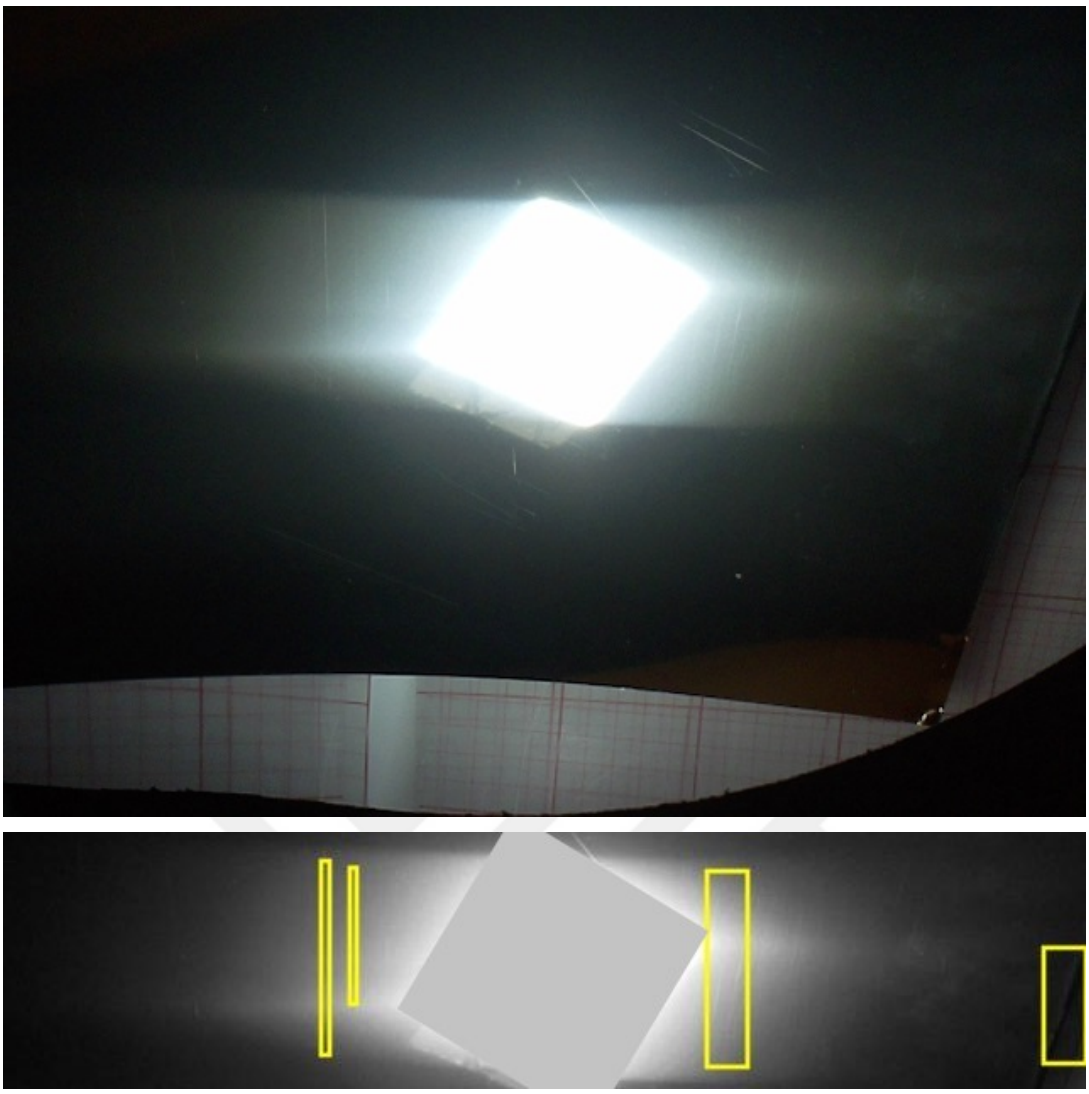
To discuss the results, sensitivity is below 20%. This means that only one fifth of the scratches were successfully found, in broad terms. This is expected, because power of this algorithm is low, in other words it is intended to detect strong scratches, defined

in Sec. 4.4.3. Moreover, parameters are tuned for images that are shot closer, than the ones in the data set. In fact, those results are well enough for the needs of customer, but fails at vague area and causes a fairly low sensitivity.

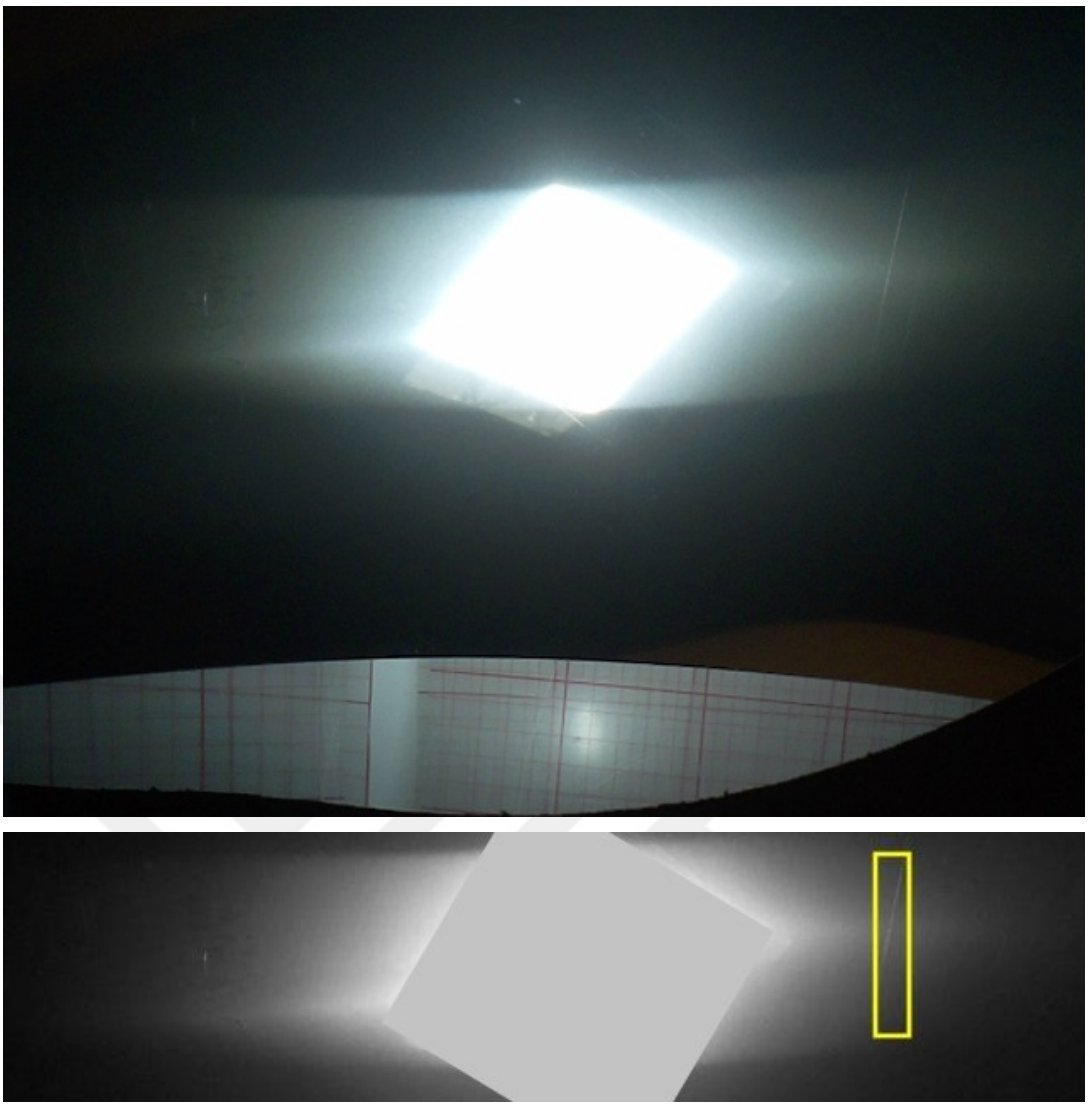
To prove our point of the results are good enough on finding many scratches, please see Figures in 4.25.



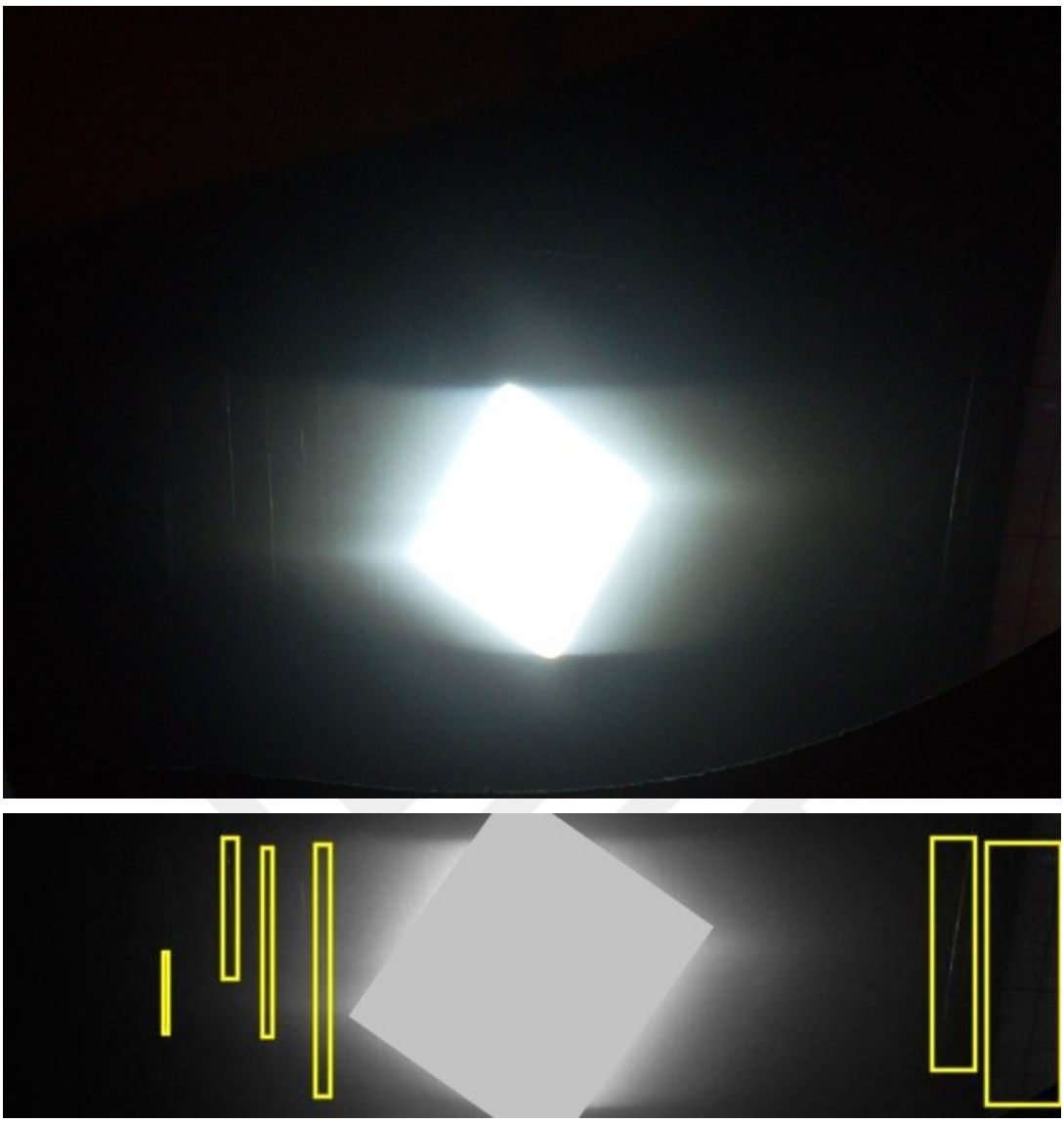
(a) Input and Rectangular Result 1



(b) Input and Rectangular Result 2



(c) Input and Rectangular Result 3



(d) Input and Rectangular Result 4

Figure 4.25: Resulting images for low-cost system modeling

4.4.3.3 Evaluation

In the figures (4.25), it is observed that the system is able to spot certain number of scratches, and those are the ones with strong features (see Sec. 4.4.3 for definition). The timing measurement values are very promising for a slow platform like MATLAB. In hardware implementation of Raspberry Pi (Sec. 4.4.2), a faster system was

expected. But, it should be noted that an image acquisition by camera is be included at that system which slows down the operation slightly.

4.5 Mid-End System

4.5.1 Principles of the Mid-End System

In this section, particularly, we will introduce brief background knowledge on theoretical aspect of mid-end solution. The focal point of this system is robustness. Robust systems are able to solve a problem under a variety of conditions successfully, or easily adapt to another problem by tuning. In this one, we have implemented a deep learning algorithm network that is convolution based.

Segmentation method applied here can be regarded under the title *supervised segmentation*. This type of approach needs an input and also a ground truth of this input. Ground truth is the user defined area for true objects on an image. Specifically, our main scope in thesis is *classification* of true objects.

A classic *classification based supervised segmentation* consists two major steps, namely, feature extraction and decision. Features may be lines, corners, colors or more complex attributes of an image object. The decision is done by machine learning, which divides the feature space into meaningful dimensions and clusters. When an object is a member of a certain cluster, then this object is announced as a classified object.

Neural Networks

Those are the type of network that we have chosen for learning task. Using the high level of operation power of recent GPUs and CPUs, a computer can solve complex mathematical problems in milliseconds. But, human brain, although being simpler in mathematics, is able solve even more complex problems thanks to the parallel structure inside. To fulfill tasks like image classification, artificial neural networks are coined in 1940s. A building block of such a network would be *artificial neurons*. Just like its biological counterpart, an artificial neuron has dendrites that are excited

by other neurons and makes an impulse signal to the output, considering a threshold [95]. A neuron function generating binary output is expressed as *perceptron* (Eqn. 4.6) [96].

$$a = \sum_{j=1}^{N} u_j w_j + \theta \tag{4.6}$$

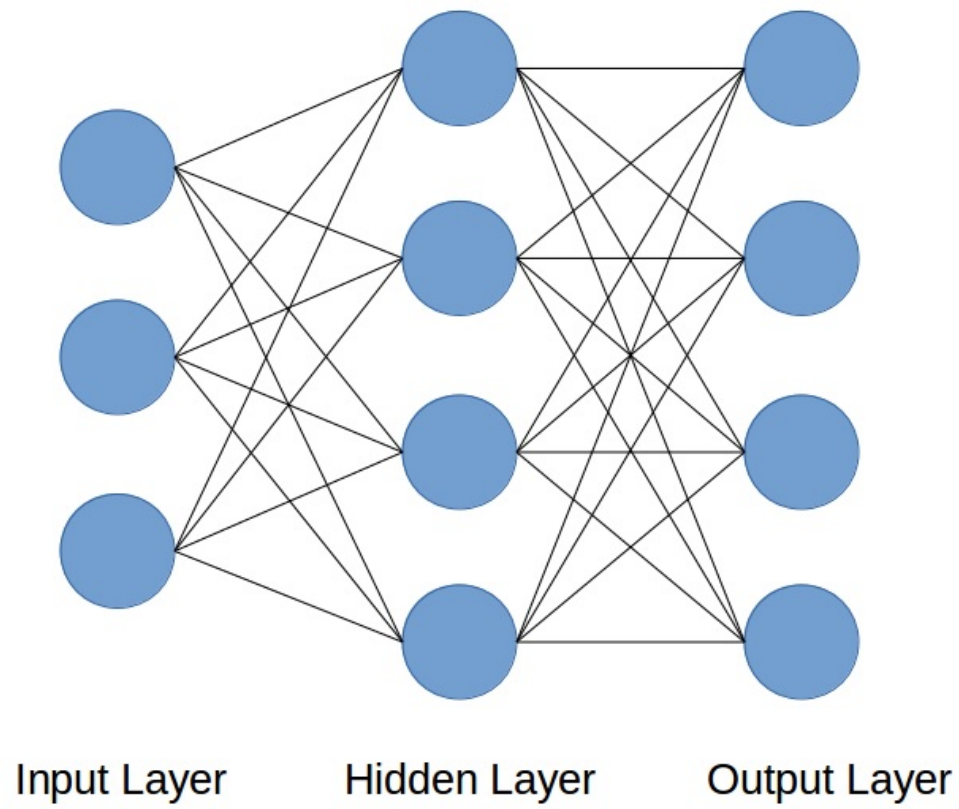


Figure 4.26: A layered network built by neurons

In above equation w represents weights, while u is an input coming from other neurons. To make it a perceptron, the activation function $f(a)$ has to be a threshold function.

Our network structure is *feedforward*, meaning signals flow through neurons in a

single direction. Those neurons form a layered network (Fig. 4.26 and if there are more than one hidden layer, then it is called a *deep neural network*.

All in all, all the neurons in the network has its own parameters for every input synapse they have. Those parameters would make matrices and the matrices will needed to be trained somehow. This is an optimization problem to be solved with backpropagation methods.

Gradient Descent

Weights are updated according to cost functions. The optimizers strive to minimize cost or errors in a network. *Gradient descent* is one the most efficient and popular methods [97].

This methods tries to find a local minimum, iteratively. At each iteration, first derivative of the function is compared with previous steps. Equation 4.7 shows the how it is derived.

$$w(i+1) = w(i) + \delta w(i)$$

$$\delta w(i) = -\eta \nabla \varepsilon(w(i))$$

$$\nabla \varepsilon(w(i)) = \frac{\partial \varepsilon}{\partial w(i)} \tag{4.7}$$

To train a network single, batch or mini batch samples can be used. Batch sample includes every single sample. Mini batch sampling that we will use, network is fed with a number of samples. Batch sampling is strong against local minima and give stable results, but for the cost of a long training time. Single sampling is not result in a good optimization. So, mini batch is the middle point between. Batch size is chosen regarding data size and hardware capabilities.

Adam Optimizer

This is a upgrade to conventional gradient descent approach. By combining advantages of AdaGrad [98] (successful at problems with sparse gradients) and RMSProp (successful at noise problems) algorithms, Adam optimizer is claimed to achieve better results. In machine learning, decreasing the learning rate as the iterations increase, is a common approach. Adam handles this by itself. It also assigns a different learning rate for each trained parameter. Additionally, Adam utilizes a moving average of the first and second momentums for updates. Another important feature of Adam is that prior to parameter update, it introduces a bias correction. This avoids large step sizes, and thus, divergence [99].

Backpropagation

There are many weights in a single network. Optimization is a complex one, could not be solved by a basic single equation. The chain rule comes in action. The error propagated from output to input gradually, that is *backpropagation* [100]. Using backpropagation, an equation is found for each weight so that there are same number of equations and unknowns. This is done by propagating error through input. Then, the derivative of those propagated errors are calculated according to the weights correspondingly. On a multilayer network, backpropagation is done by chain rule application.

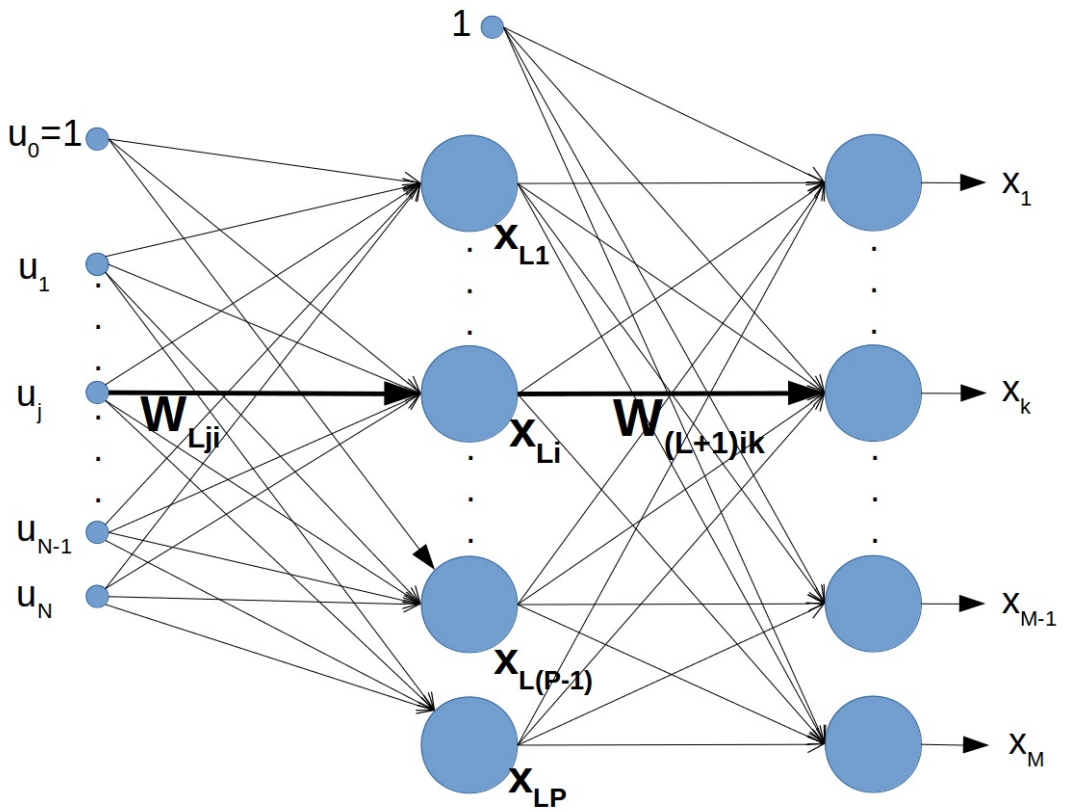


Figure 4.27: A simple deep network with weights indicated on

Examining Figure in 4.27, starting from the output layer chain rule is applied as in Eqn. 4.8.

$$x_k = f(W^T_{(L+1),k} x_L)$$

$$x_{L,i} = f(W^T_{L,i} u)$$

$$\frac{\partial \varepsilon}{\partial \omega_{L,j,i}} = \frac{\partial \varepsilon}{\partial x_k} \frac{\partial x_k}{\partial x_{L,i}} \frac{\partial x_{L,i}}{\partial \omega_{L,i,j}}$$

$$\frac{\partial \varepsilon}{\partial w_{L,j,i}} = -\frac{2}{M} \times \sum_{k=1}^{M} (t_k - x_k) f'(W^T_{(L+1),k} x_L)(w_{(L+1),i,k}) f'(W^T_{L,i} u)(u_j) \quad (4.8)$$

The error ε is chosen to be *mean square error* for above derivation. To apply gradient descent, $\frac{\partial \varepsilon}{\partial \omega_{L,j,i}}$ is needed. Starting from first output neuron to Mth with the index k, error change with respect to weight is calculated for each neuron. Activation functions $f()$ varies and depends on the designers choice.

From this point on we will discuss about layers of a CNN structure. The aspects introduced in this section so far, are implemented in every CNN. But, the remaining are not must.

Convolution Layers

Those are of course the main type of layer in a CNN. The convolution layer actually corresponds to a special type of *feedforward* network layer, where each pixel p_{ij} corresponds to an input u_j and each value w_{ij} of the convolution filter corresponds to a weight w_j.Assuming a filter size of 3×3 and an input size of $d \times n$, there are $(d-2) \times (n-2)$ neurons with $(d-2) \times (n-2)$ number of outputs. Unlike a traditional network, which would have $(d-2) \times (n-2)$ number of different weight matrices, there is only one weight matrix that is shared among all neurons in a convolution layer. This is the most important characteristic of a convolutional neural network. What makes shared weights possible is that the features do not change according to spatial positions.

The transfer function Equation 4.6 corresponds to a convolution in a CNN. A simple 2D convolution with 3×3 weight matrix is formulated in Equation 4.10 (Note that this equation does not take the symmetric of the filter unlike the general 2D discrete convolution formula in Equation 4.9). Basically, one may refer to a convolution filter as an artificial neuron given in Figure 4.29. This operation is applied on whole image

by sliding the filter as shown in Figure 4.28.

$$C_{ij} = \sum_{k=-1}^{1} \sum_{l=-1}^{1} A_{(i-k)(j-l)} B_{(k)(l)} \tag{4.9}$$

where $1 < i < D$ and $1 < j < N$ and dimensions of matrix A is $D \times N$

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & p_{1n} \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{d1} & p_{d2} & p_{d3} & \cdots & p_{dn} \end{bmatrix} \qquad W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

$$R = \begin{bmatrix} r_{22} & r_{23} & r_{24} & \cdots & r_{2(n-1)} \\ r_{32} & r_{33} & r_{34} & \cdots & r_{3(n-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ r_{(d-1)2} & r_{(d-1)3} & r_{(d-1)4} & \cdots & r_{(d-1)(n-1)} \end{bmatrix}$$

$$r_{ij} = \sum_{k=-1}^{1} \sum_{l=-1}^{1} p_{(i+k)(j+l)} w_{(2+k)(2+l)} \tag{4.10}$$

where $1 < i < d$ and $1 < j < n$

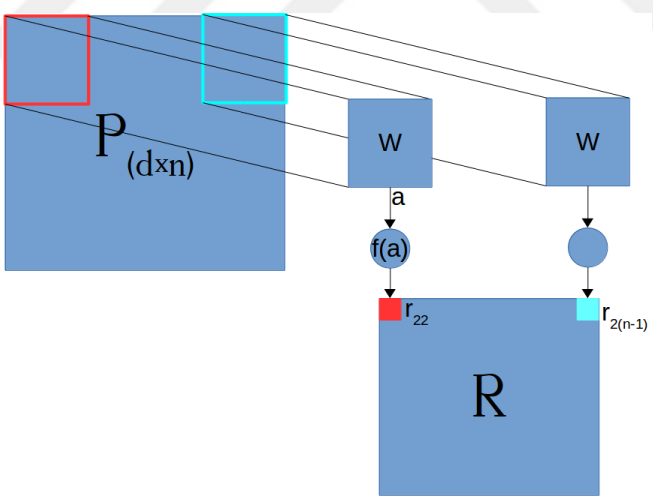


Figure 4.28: Convolution is applied on the entire image.

The red rectangle in P represents 3×3 pixels on the upper left corner. The result is the red rectangle in R and it is actually r_{22} in Figure 4.29.

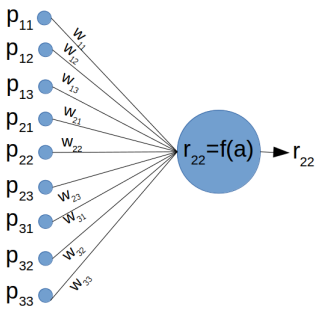


Figure 4.29: A neuron representing the filter W applied on pixels around p_{22}.

If there are multiple channels, then the filter W becomes a 3D matrix with the same number of channels.

In other words, convolution layers consist of filters that will be applied on each channel of the input. Since applying a filter on an image is basically a convolution, these layers are named convolution layers. By applying filters, features of an image that correspond to each filter is found. Similar to the architecture proposed in [101], as the layers go deeper, the features become more complex. For example, assume that in the first layer, vertical and horizontal edges of an image are found. In the second layer, the outputs of the first layer, which correspond to vertical and horizontal edges of the image, are used to find the corners in the image. Of course in real applications the features get much more complex in the deeper layers of a network.

Pooling Layers

Pooling layers are usually used to reduce the dimensions of the output of a convolution layer. These layers are needed because number of features obtained from a convolution layer can be undesirably high. For example, assuming zero padding on the edges so that the convolution result has the same size with the image, an RGB image of size $H \times W \times C$ would have $H \times W \times (number\,of\,filters\,in\,the\,layer)$ features. Training on such high number of features would result on problems like overfitting. Therefore, dimension reduction is needed. There are many methods that can be used to reduce the dimensions, such as average pooling and max pooling. The

most preferred one is max pooling because it extracts the feature that is dominant in that local region where the pooling is applied. This is particularly critical since this module allows to propagate the error with the highest responses so that it decreases the chance of vanishing gradient problem which might be frequently observed for the averaging operations. Figure 4.30 shows max pooling and average pooling applied on 2×2 windows with stride 2.

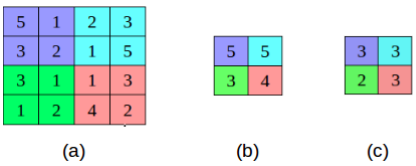


Figure 4.30: (a) Image pixel values. (b) Result of max pooling for each colored region. (c) Result of average pooling for each colored region.

Upscaling Layer

Not always downsampling is needed in neural networks. Upsampling is also needed in neural networks for tasks like pixel-wise classification, or semantic image segmentation. After reducing the spatial dimensions as in classical classification task, a CNN built for pixelwise classification needs to construct an output with the same spatial dimensions of input. Thus, upsampling layers are used to increase the dimension of the deep features. There are many methods used as an upsampling layer in CNNs. Bilinear interpolation, deconvolution and subpixel upsampling are some of the important ones.

Interpolation

Bilinear interpolation is the simplest method to upscale a 2D image. It is an extension of linear interpolation. In linear interpolation, the effect of a known point, on the value of the target point is inversely proportional to the distance between them. The equation of the linear interpolation is given in Eqn. 4.11. It actually finds a point on

a line, given one of its coordinates and the equation of the line.

$$y_3 = y_1 + \frac{(x_3 - x_1) * (y_2 - y_1)}{(x_2 - x_1)} \tag{4.11}$$

Bilinear interpolation extends this equation to 3D space, considering the 2D coordinates and the pixel values. It can be seen in Figure 4.31

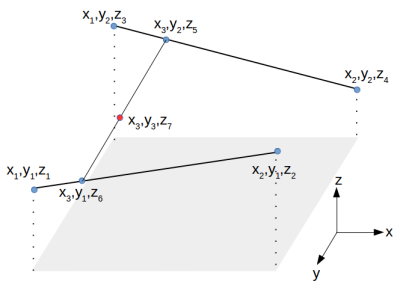


Figure 4.31: Bilinear interpolation

Fully Connected Layers

Fully connected layers use the feature information to reach a final class label. Nodes of fully connected layers are connected to each output of the preceding layer and their outputs are connected to each node of the following layer. In the last layer, there are n number of nodes, where n equals to the number of classes. The nodes in the last layer of the fully connected layers generally have a special activation function that will generate a probabilistic result for each label. This activation function is generally a softmax function, or a sigmoid function in binary classification case.

1x1 Convolution

If the aim of the network is to perform segmentation, it is a fully convolutional network. The dense layer at becomes a 1×1 convolution layer with a different activation function. In the classical classification problem, the output layer is a vector with a single dimension equal to the number of classes. In the pixelwise classification problem, on the other hand, the output is a 3D matrix. Its width and height is equal to those of

input image while its depth is equal to the number of classes. So, one might say that there is an output vector for each pixel in the image, resulting in an output of dimension $Width \times Height \times NumberOfClasses$. Particularly, one can represent a fully connected layer in a classifier by 1×1 convolutions applied to single pixel images. In this case, the output's dimensions are $1 \times 1 \times NumberOfClasses$.

After the input of an artificial neuron is multiplied by the weights and summed to a scalar, the result is fed into an activation function $f()$. In this section, different activation functions will be given and their advantages and disadvantages will be discussed.

Softmax

Softmax function (Equation 4.12) is usually used in the output layer of multilayer neural networks that are built to do classification task. It actually calculates the probability of each class. Since we are speaking of probabilities, the sum of all softmax results in an output layer equals to 1. Instead of making a strict choice between classes, softmax estimates which class the input more likely belongs to. Assuming there are M classes, softmax can be represented as:

$$softmax(x_k) = \frac{e^{x_k}}{\sum_{i=1}^{M} e^{x_i}} \tag{4.12}$$

Rectified Linear

Until publishing of [102], it was believed that differentiable, non linear, symmetric functions were better to use in neural networks. However, [102] has shown that rectified linear units(ReLU) makes training a neural network computationally more efficient and faster. Due to its being differentiable everywhere except 0, it is easy to calculate the gradient descent (Eqn. 4.13) for ReLUs. At 0, on the other hand, its derivative is accepted and defined by the user as either 0 or 1. Additionally, since the function does not saturate, the gradients do not vanish for very large values. Therefore, the training is faster.

$$ReLU = max(0, x) \tag{4.13}$$

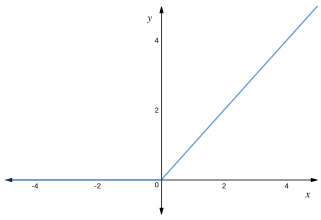


Figure 4.32: ReLU function

ReLu Variations

Rectified linear units somewhat ignores the negative input values. Any negative valued input results in a 0 output and the gradient is 0. In order to keep the constant gradient and to avoid the dying problem, leaky rectified linear units (LReLU) are used. In LReLUs, the function becomes Eq. 4.14

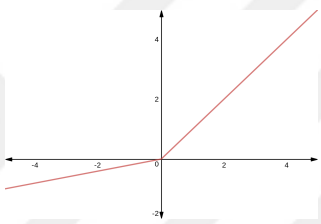


Figure 4.33: LReLU function as given in Eq. 4.14(where α is 0.2)

$$LeakyReLU = max(\alpha x, x) \tag{4.14}$$

where α is a positive constant smaller than 1. Note that if α is trainable value, the function is called *parametric rectified linear unit (PReLU)*.

Batch Normalization

Since the datasets used in deep learning are huge, the input of network varies significantly. Batch normalization is, as the name suggests, normalizing the data in the mini batch and making the normalization part of the network, instead of normalizing the whole dataset as a pre-processing method. Thus, in tasks like image classification, it increases learning speed [103]. Additionally, [103] states that the need for dropout layers may be excluded since batch normalization acts as a regularizer.

4.5.2 Implementation of Mid-End System

In this section, an overview of the proposed mid-end system will be given. This system consist of a mechanic frame, a light source (more can be added if needed) and a *Nvidia JETSON TX2* a powerful computer that is upgraded with a *Standard Camera Module*, comes with development kit. The resulting system is expected to be faster compared to low-cost one, to return more accurate image segmentation, robust and fast enough to be a real-time solution.

The algorithm is coded mainly in Python 3.5 language, and it was chosen almost as a must, which is due to the fact that deep learning related frameworks commonly runs in Python. Tensorflow was the chosen framework, as it is very flexible and good at handling background operations for such problems. The greater advantage was that transferring a network model from PC to Jetson is quite easy, when both devices have Tensorflow environment installed. Image acquisition is managed by a streamer library of Python by Nvidia. In this section, a brief background knowledge will be introduced, and moving onto implementation image acquisition and the model designed will be discussed.

4.5.2.1 Image Acquisition

Image acquisition, in contrast with low-cost system, was not deeply studied for this system. Although, its camera has a fewer megapixel sensor than Raspberry's, Jetson is capable at transferring high quality images with low noise into the main processing board. There was less need in fine adjustment of camera specifications. Also, motion blur was not experiences as we did for low-cost system.

Resembling to *picamera* firmware interface, Jetson camera can be commanded by *nvcamerasrc* framework. It is a Gstreamer pipeline designed for Jetson cards created by Nvidia itself [104]. This firmware interface makes a connection to ISP unit of the graphical units, converting Bayer data into YUV and other types.

Unlike studied for the low-cost system in Section 4.4.2.1, there was a shallow practice was done on TX2 camera module to acquire images. Because, the card was power-

ful enough to process video images with high quality and FPS. Although, we may say that Raspberry Pi camera is better in a general sense, in comparison with Jetson Development Kit camera.

Still, we should discuss some of the properties assigned for the camera module.

Sensor Resolution 2592×1944

Capture Format I420

Acquired Resolution 640×480

Acquired Format BGR

White Balance Auto

Color Effects Off

Auto Exposure On

Contrast 0

Exposure Time $0.33\ sec$

Edge Enchancement 0

FPS 30

Those are some important parameters and majority are in their default options. Bayer sensor resolution is chosen as 2592×1944, that is a high value, but it is a capability of Nvidia Jetson to handle such high resolution frames. But, in ISP, all images are converted 640×480 resolution and acquired with at least 30 FPS. The input images comes in BGR colorspace, namely blue-green-red. There is no strong denoise filter in the camera module hardware, acquired images are denoised with software tools, instead. By the way, exposure time is not applicable for video operations, otherwise, default FPS would be around 3.

4.5.2.2 Algorithm and Hardware Results

This section includes depiction of deep neural network model and also visual outputs from a real-time session as in Section 4.4.2.2. Actually, what time consuming is in this implemention was preparing the software medium. Because, design and also coding have been done during modeling and explained in Section 4.5.3.

In literature, there are some small neural network models like Tiny YOLO, Mobile SSD, or Squeezenet [105]. But even those consist of at least 8 convolutional layers and they are intended to segment more complex objects like cat, dog, car, human etc. As our main aim is to detect linear scratches, it was decided to propose a simpler neural network architecture. The convolution deep neural network designed consist of 4 main convolution blocks (Fig. 4.34), maxpooling after first three, and upsampling in the end with a 1×1 convolution layer. Whole architecture is demonstrated in Figure 4.35.

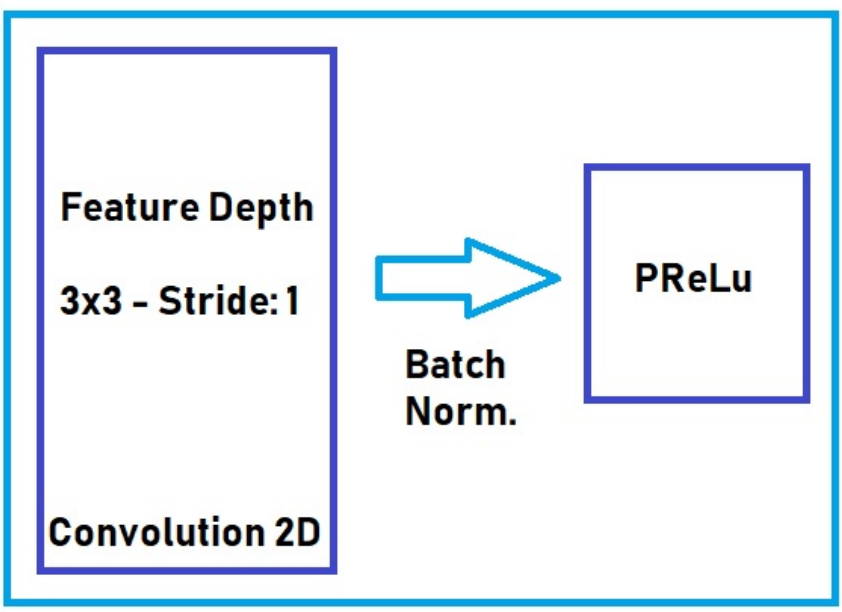


Figure 4.34: Operations in a single convolution block

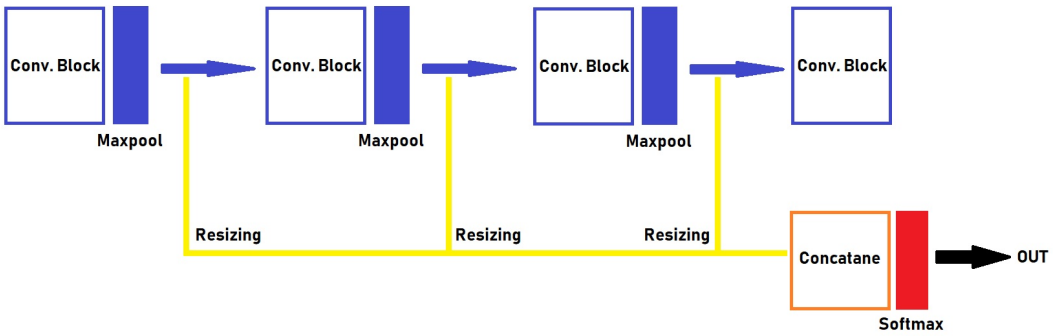


Figure 4.35: Whole CNN architecture for mid-end system

Each convolution block starts with a 3×3 convolution operation with 1 pixel stride. The number of filters depend on the parameter *feature depth*. This parameter is the main one changing model size and timings. In Section 4.5.3, this parameter is changed in 64, 128 and 256 numbers. But both model nominated for hardware side are made with 256 feature depths, Tab. 4.12.

Here are the convolutional filter numbers for each tensor channel, given in Table 4.7.

Table 4.7: Number of convolution filters per tensor channel in blocks

Block Number	Filter Count
1	32
2	64
3	128
4	256

After each convolution batch normalization, parametric rectification and 2×2 max-pool are applied in order. All of them are theoretically introduces in the previous section. In the end of 4 convolution blocks, upsampling is done by bi-linear interpolation, which also functions as resizing. All tensors are resized to original image size

and concatenated. This 3-channel matrix is filtered with 1×1 convolution layer. The soft results are obtained with a final softmax operation.

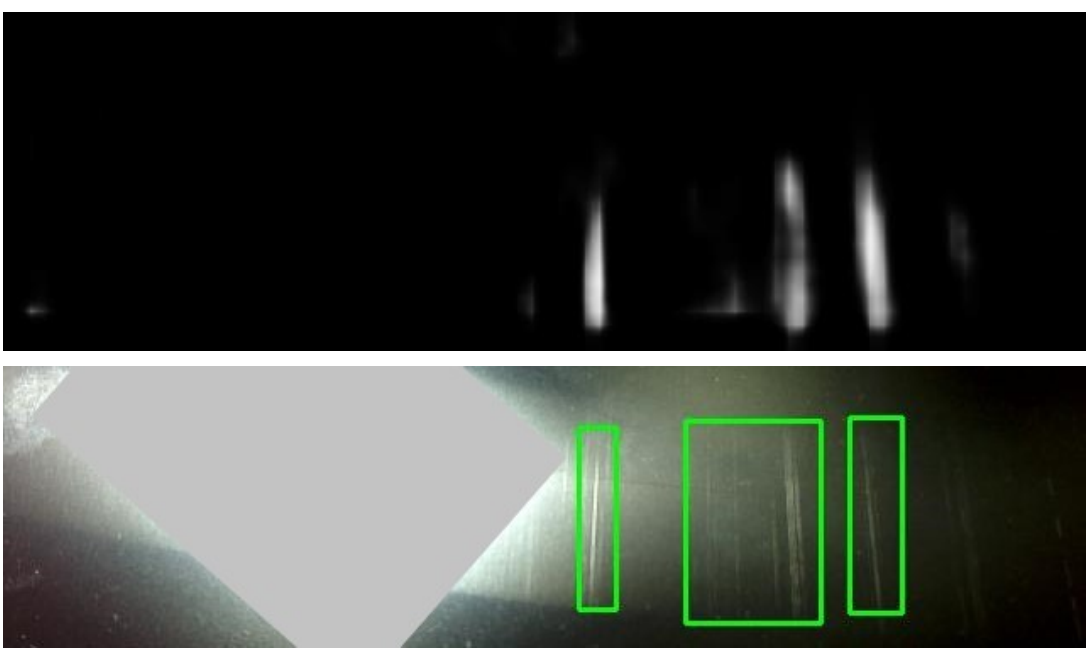
Algorithm 2 Algorithm for Mid-End Method

```
Library Import
< Time Tick 1
Parameter ← CorrespondingValues
MemoryAllocation
procedure WITH TENSOR SESSION
    if Training = 0 then
        Restore Model
    else
        Initialize Model
    end if
    < Time Tick 2
    while True do
        < Time Tick 3
        Capture Image
        Gaussian Blur
        Light Source Detect
        CNN Segmentation                    ▷ (see Fig. 4.35)
        Result Showing                      ▷ Optional
        < Time Tick 4
    end while
end procedure
```

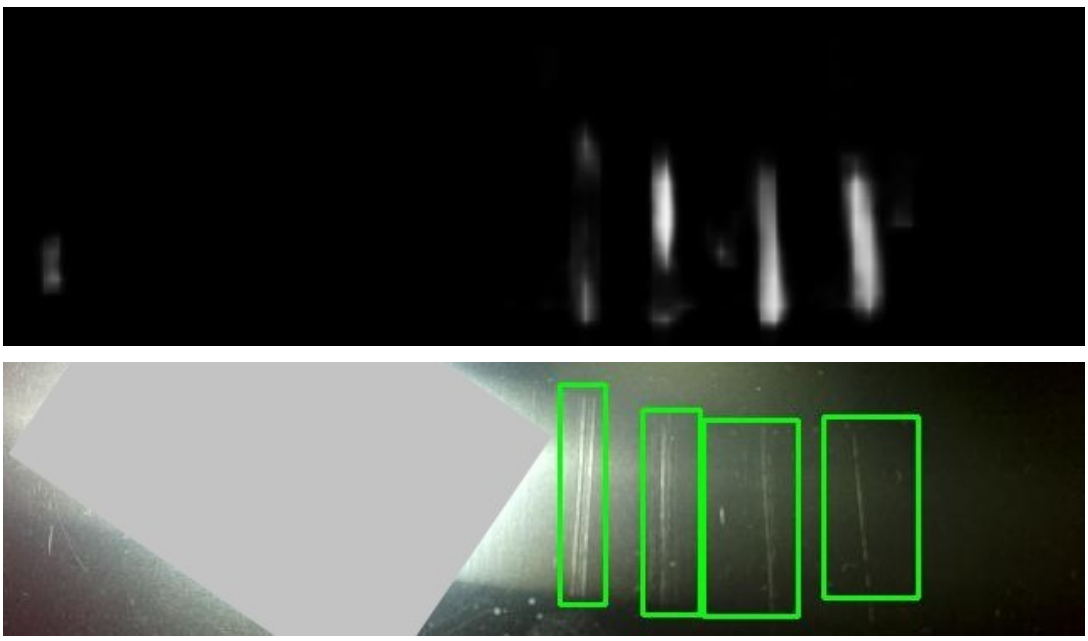
Algorithm, first, starts with introducing libraries and adjusting parameters, like *feature depth*, *Gaussian kernel size, height and soft filters*. Tensorflow framework functions with allocating everything in the beginning and starting a session for implementing those allocated units. In session, model is either restored or saved depending on it is a training or test run. Gaussian blur is applied with a 7×7 kernel. Finally, images goes through custom network of ours (Fig. 4.35), soft results are thresholded with *soft filter* of 10 and rectangles with less than *height filter* (50 pixels in this case).

This network model is implemented in modeling (Sec. 4.5.3) and also in the following real-time algorithm, run on Jetson TX2. Two parameters sets chosen for modeling (Model A and B) are given in Table 4.12.

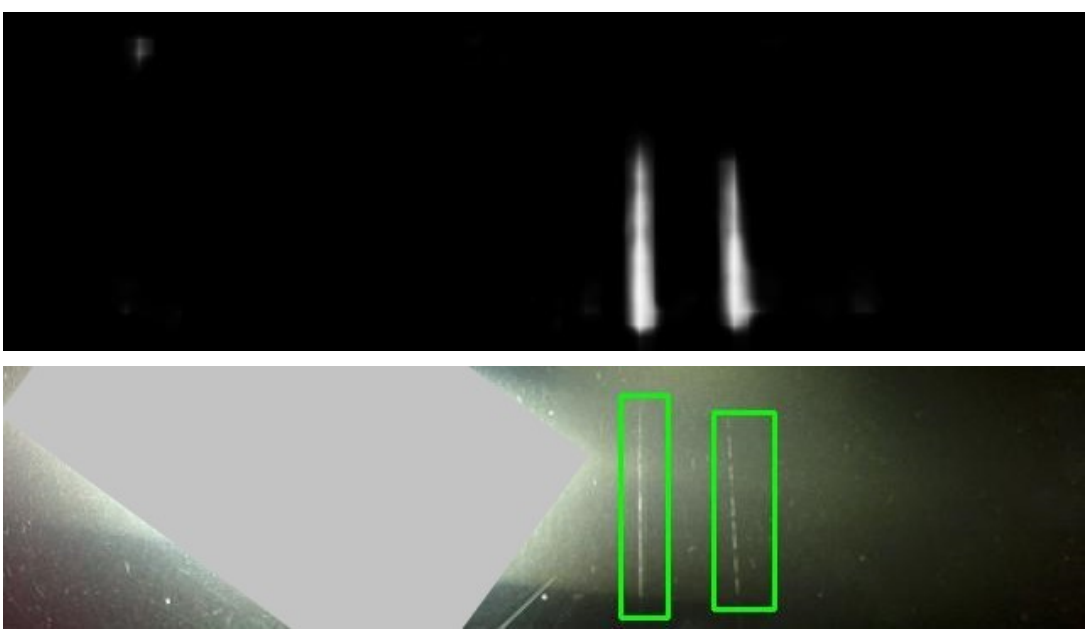
As long as this is a hardware implementation in practice, it is not possible to generate a ground truth. Because, the input images are, in fact, real-time captured frames. Consequently, it is not possible to discuss sensitivity, specificity and other segmentation metrics in this case. The evaluation will be on timing and intuitive observation on the resulting images. In the following figures, the soft results and spotted scratches with bounding boxes are given for both Model A (Fig. 4.36) and Model B (Fig. 4.37).



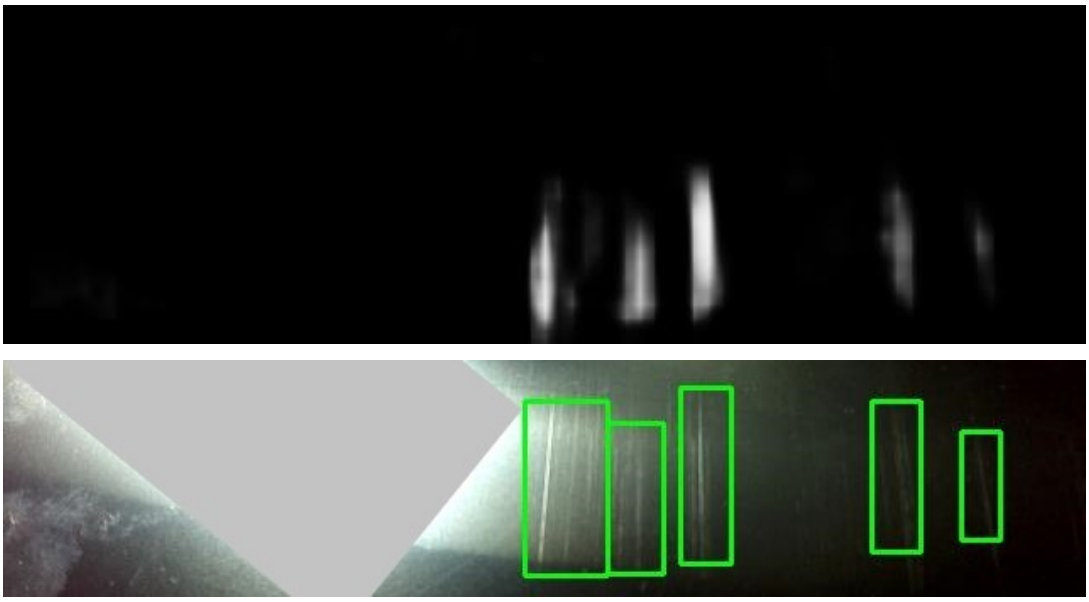
(a) Soft and Rectangular Result 1



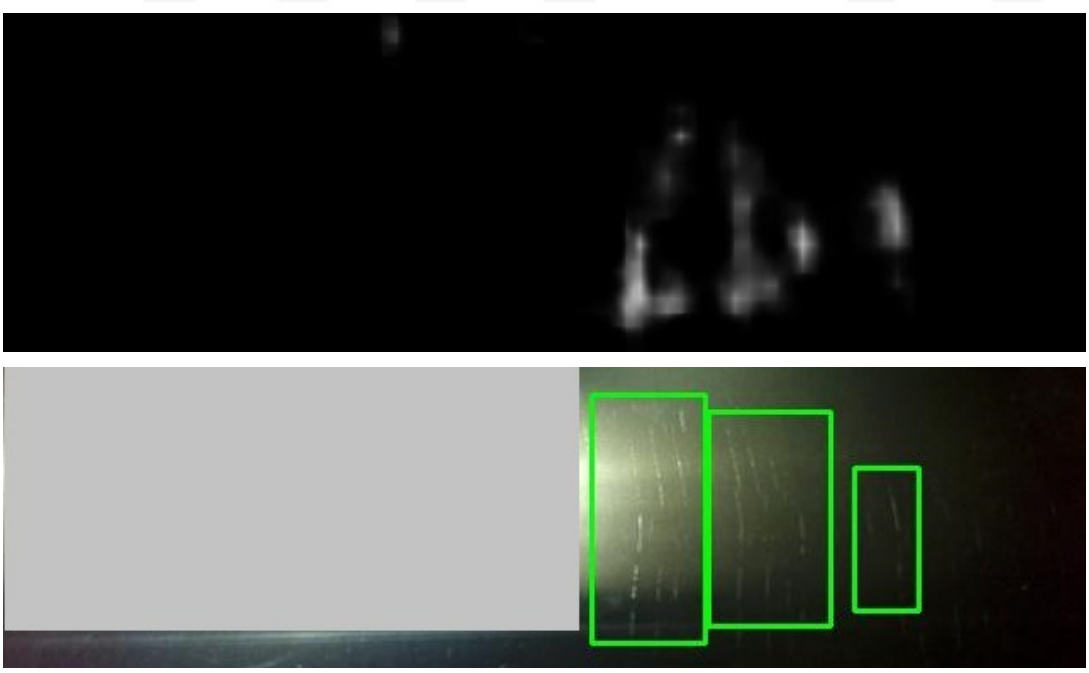
(b) Soft and Rectangular Result 2



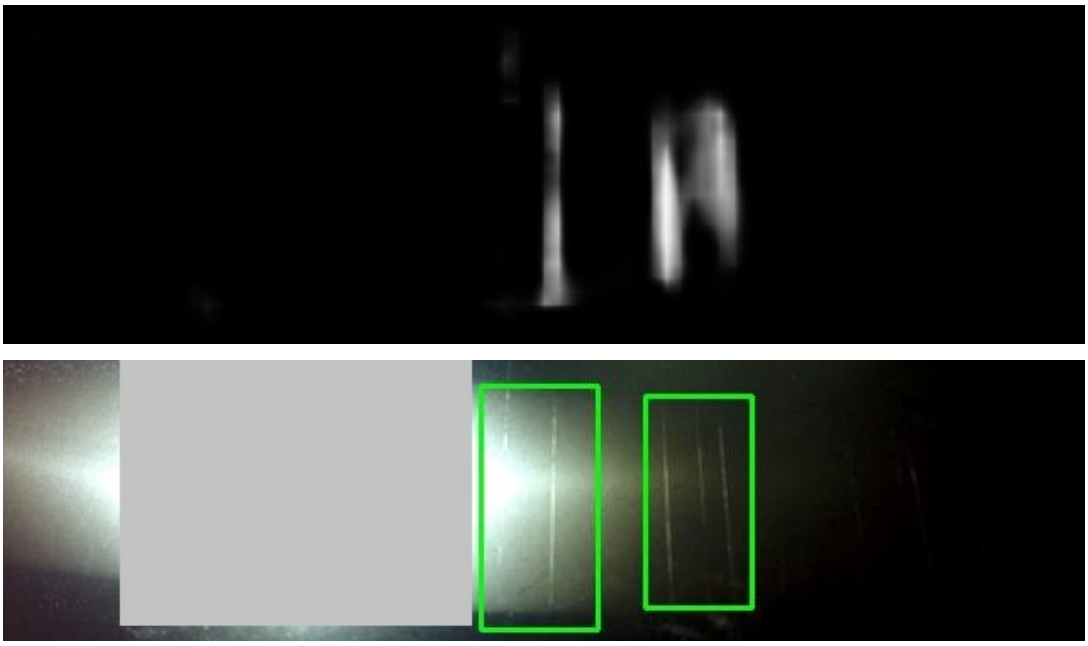
(c) Soft and Rectangular Result 3



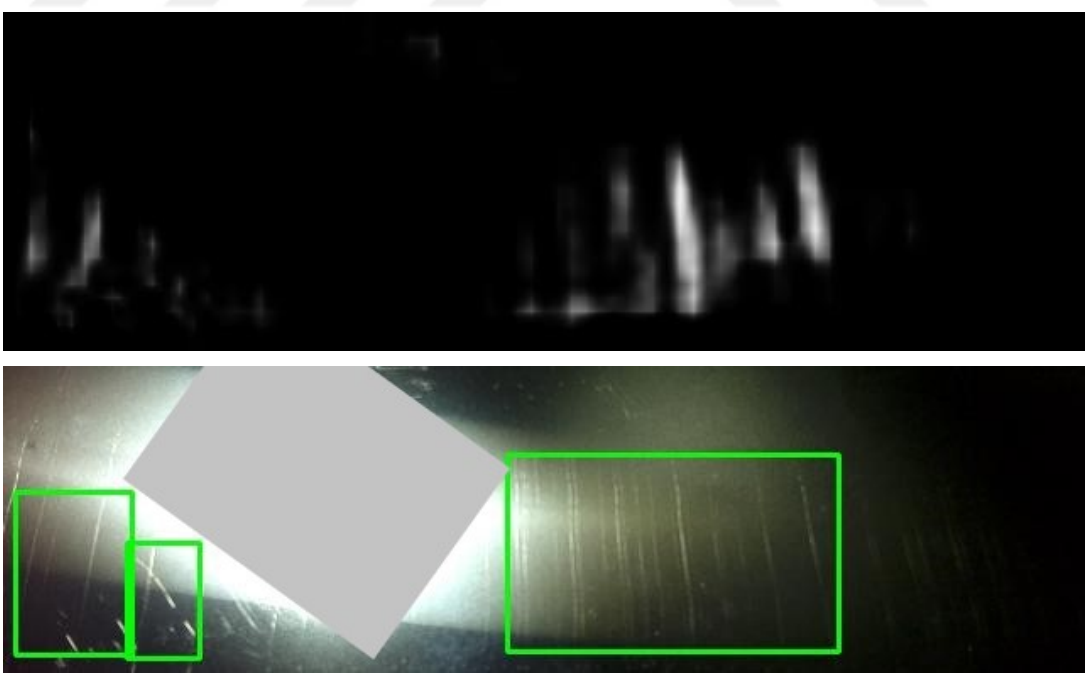
(d) Soft and Rectangular Result 4



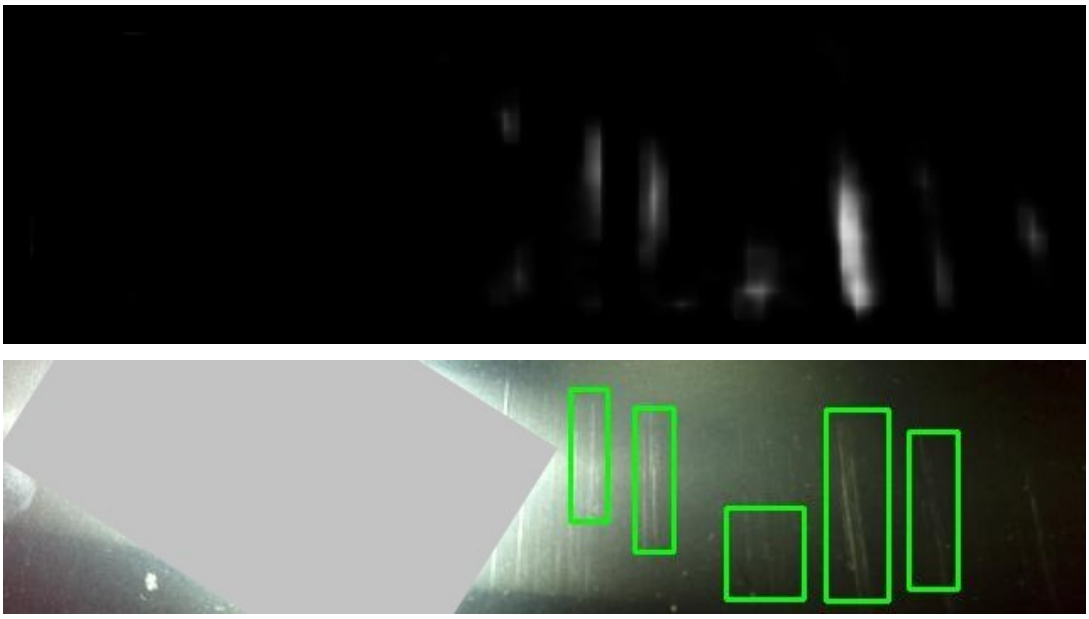
(e) Soft and Rectangular Result 5



(f) Soft and Rectangular Result 6



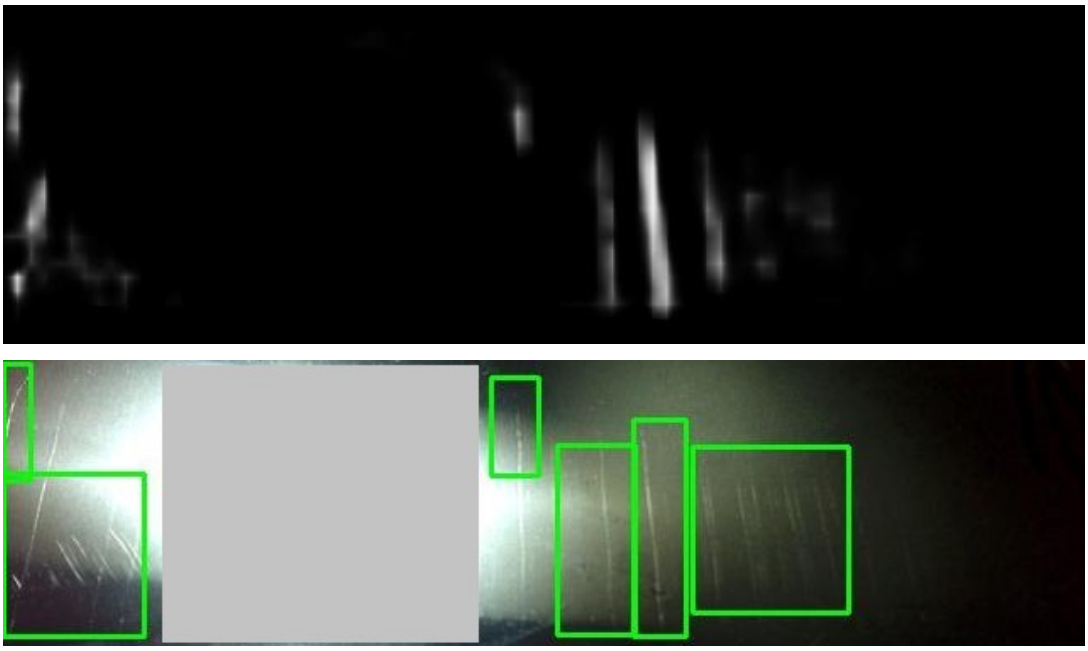
(g) Soft and Rectangular Result 7



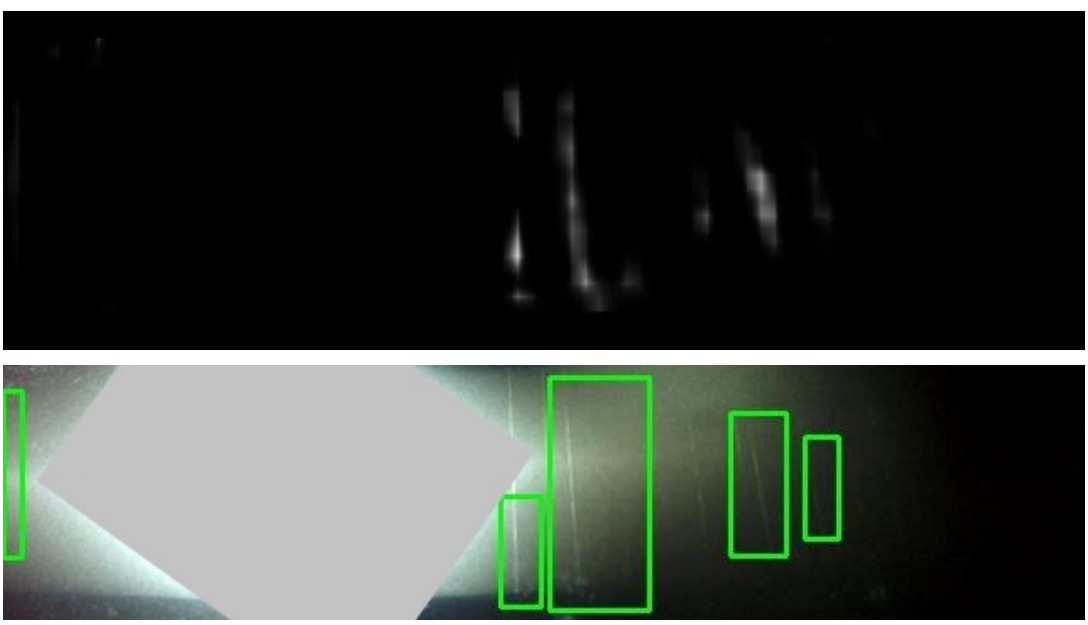
(h) Soft and Rectangular Result 8

Figure 4.36: Resulting images for mid-end system hardware implementation (Model A)

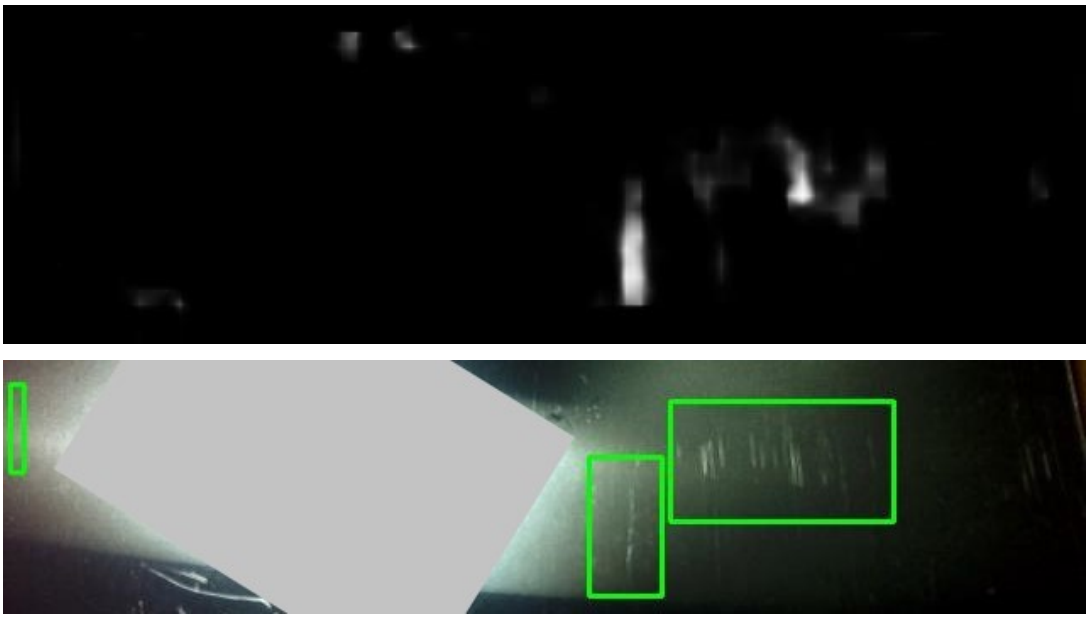
The results above presented in Figure set 4.36, all in all Jetson TX2 performance on finding scratches with such a simple network seems reasonable. In some cases, scratch clusters are considered to be one object like in Fig. 4.36a-c-d. This is due to soft results are thresholded with *soft filter* parameter, and if the blobs are connected in neighborhood, somehow, everything is merged to be single object. Model A seems to fulfill the task successfully enough, in the same way, it had high sensitivity and specificity in modeling experiments (Table 4.14).



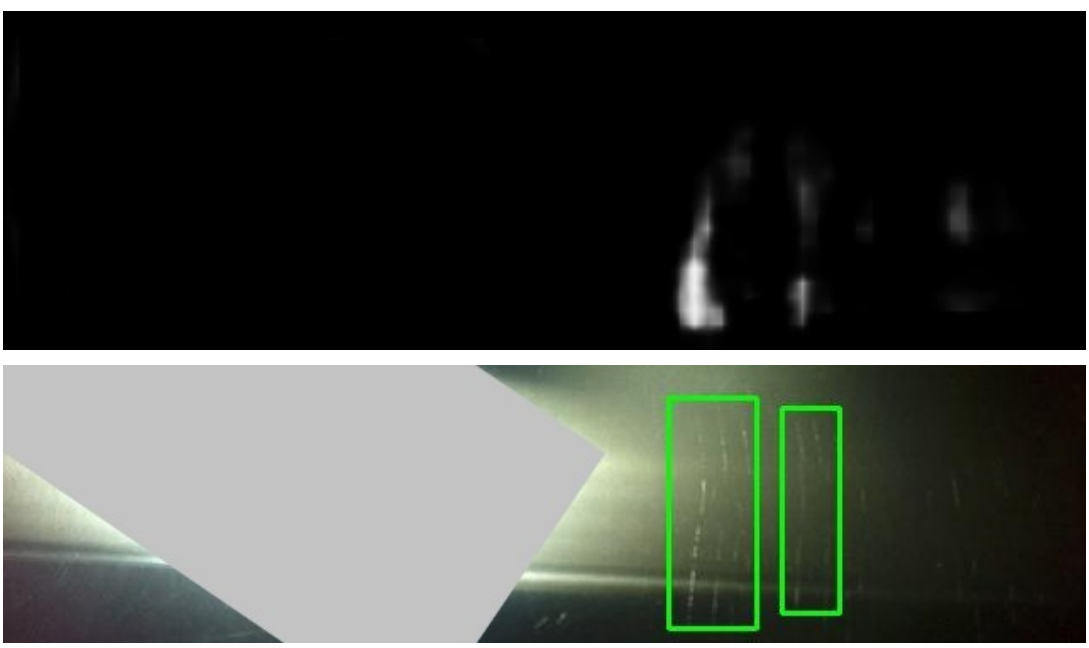
(a) Soft and Rectangular Result 1



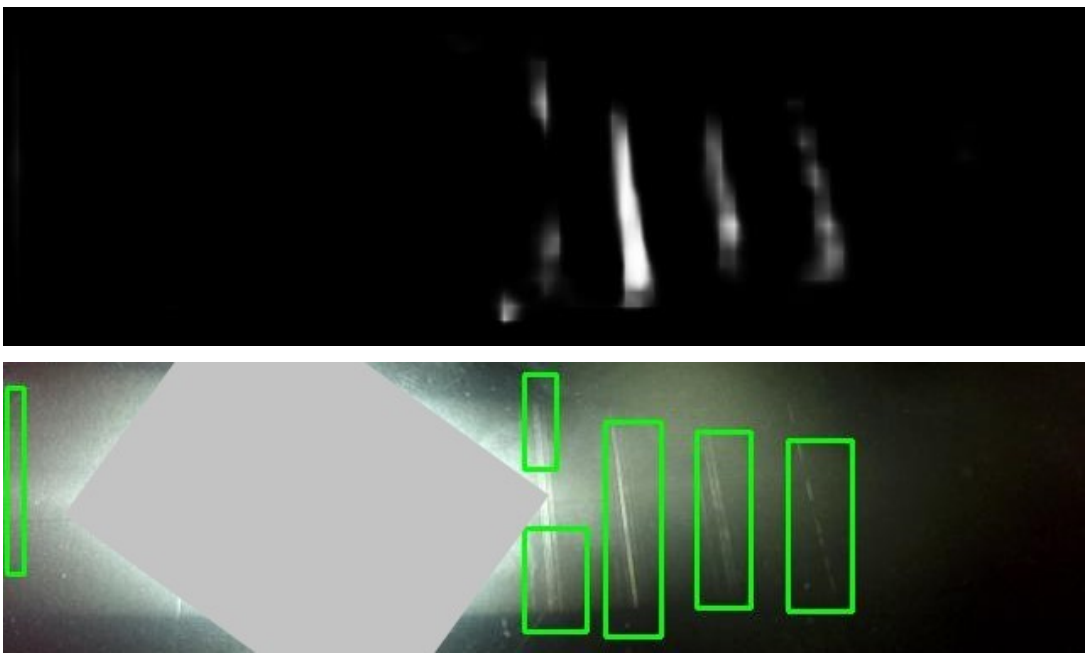
(b) Soft and Rectangular Result 2



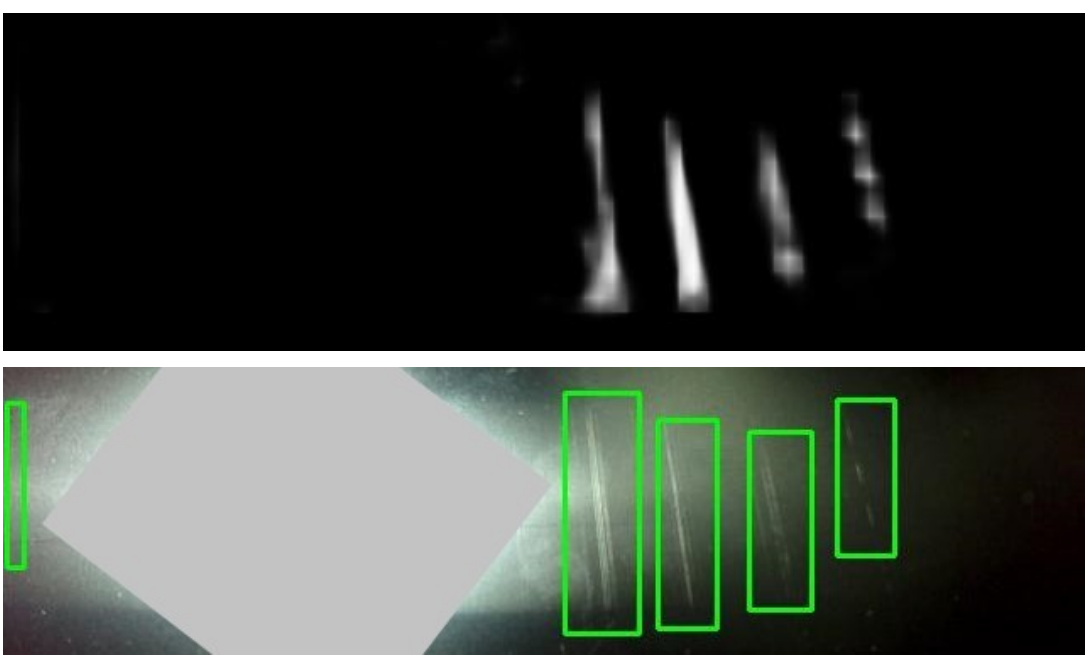
(c) Soft and Rectangular Result 3



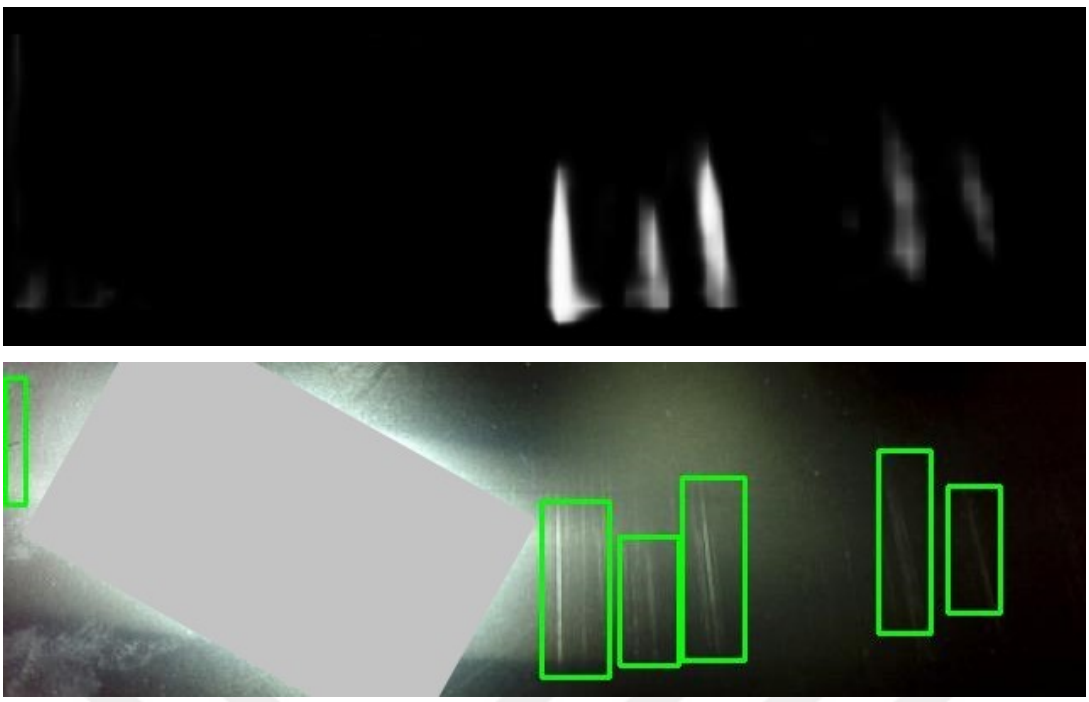
(d) Soft and Rectangular Result 4



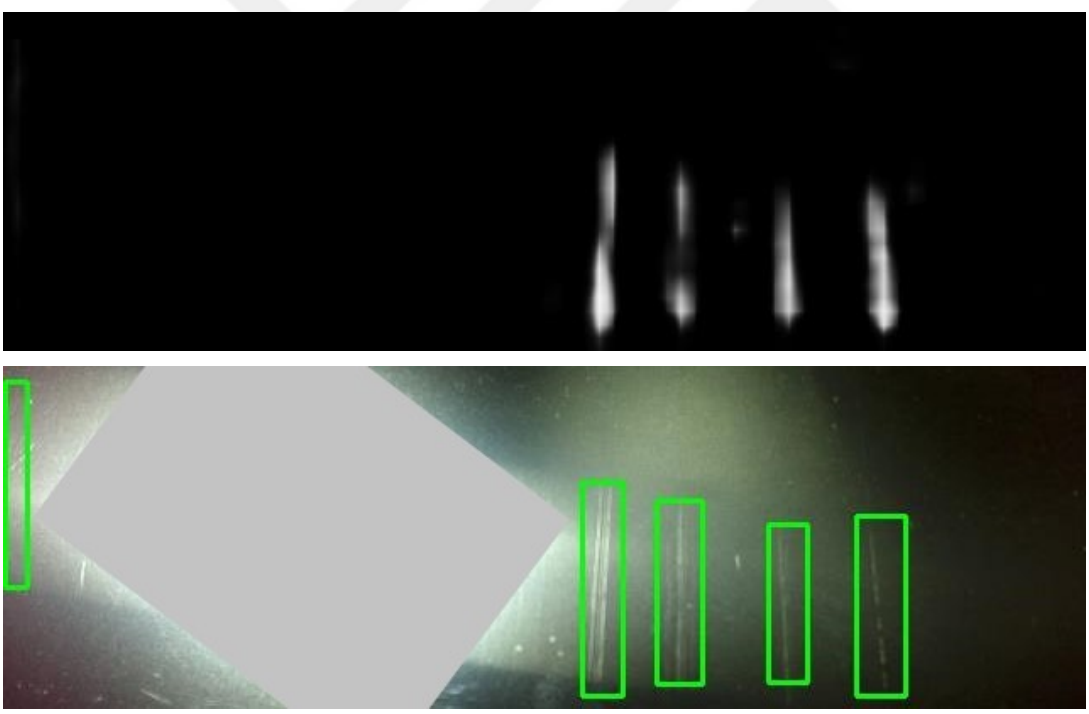
(e) Soft and Rectangular Result 5



(f) Soft and Rectangular Result 6



(g) Soft and Rectangular Result 7

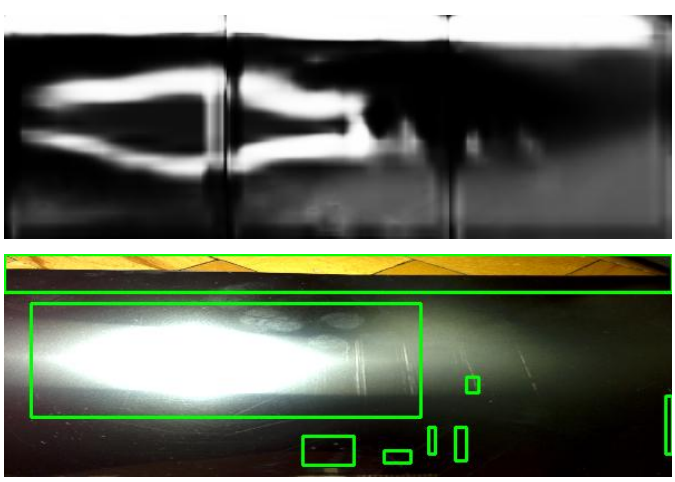


(h) Soft and Rectangular Result 8

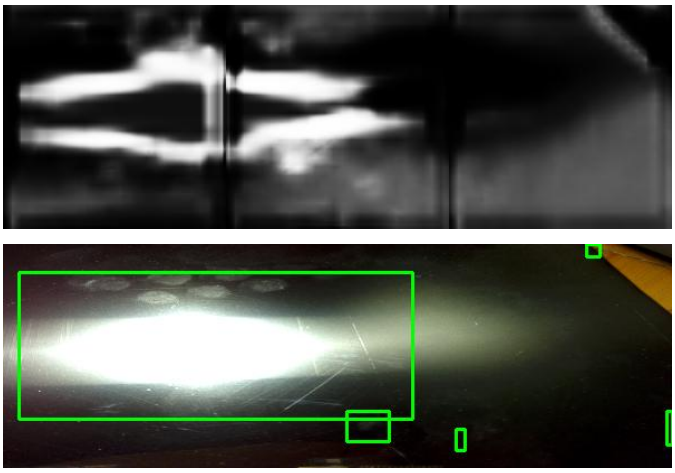
Figure 4.37: Resulting images for mid-end system hardware implementation (Model B)

Just like its counterpart, Model B was also successful on spotting linear objects in our real-time images. This model had a slightly better loss metric, meaning we expected sharper linear shapes in the soft results. Figures 4.37e-f-g-h are proof of the fact.

In addition to those models trained by Hard Set, we also made a smaller experiment with the model of which training set is NEU Set. As expected real-time images we had was not suitable with NEU Set. Because, NEU set images consist of microscopy and close-up shots. There was a subtle problem of image size in this run, due to the fact that model was trained with 200×200 images, but real-time photos can not go as small as that. The solution was taking 600×200 images, cutting it into three and feeding the system with 3 images in batches. Here in Figure 4.38, results are presented. As expected, light source is predicted as a scratch.



(a) Soft and Rectangular Result 1



(b) Soft and Rectangular Result 2

Figure 4.38: Real-time experiment of model trained with NEU data set on Jetson TX2

4.5.2.3 Conclusion

To sum up, both models trained with Hard Set or a data set with correct proportion and image size, handle the job successfully. The CNN architecture designed here is simple, yet achieving good results. Jetson TX2 was really a necessary, as it can run such many number of convolution in reasonable time. In Table 4.8, FPS values are given depending on the *feature depth* parameter. As presented in Table, FPS values are quite high and definitely whole operation is considered as real-time.

Table 4.8: Feature Depth effect on FPS

Feature Depth	FPS
256	$9-10$
128	$14-15$
64	~ 30

Time measurement are with putting time ticks in proper places. In Table 4.9, namings are indicated with which ticks. Prepare time is around 7.5 seconds and almost

the same for every different model run.

Table 4.9: The time ticks used for each time measurement

Name	Ticks Used
Prepare Time	$(T_2 - T_1)$
Loop Time	$(T_4 - T_3)$

4.5.3 Simulation Results for Mid-End System Algorithm

For modeling mid-end system, Tensorflow framework was chosen as it is very flexible for learning based designs and also handles a lot of operations automatically in behind [106]. The modeling platform and dependencies of mid-end system design are *Tensorflow v1.6, OpenCV v3.4.2, CUDA v9.0, cudNN v7.04, and Linux OS v16.04*.

On the given software medium, we coded with Python 3.5.2 version. Tensorflow takes care of CUDA features. CUDA is another language, that results in a firmware controlling software for parallel processing on a graphic cards. Parallel processing is the necessity for learning tasks for 2-D array problems, as there are logical and algebraic operations in high numbers. The cudNN is a library dependency of Nvidia which is high-performance building blocks for applications such as convolutions, activation functions, and tensor transformations [107].

There are 16 models trained in this thesis work with chosen parameters. All the models have the same layer design, but they differ in batch size, training iterations, data set and feature depth. Some of the results are gathered in training sessions and others in test runs.

4.5.3.1 Timing Results

The first concern was on the timing performance while designing the models, as we needed a model that could work in real-time. Real-time ability is not the case for deep

learning models, majorly. Here is timing results are given for Hard set and batch size of 8 in Table 4.10.

Table 4.10: Timing results for deep neural network models (Hard Set, batch size:8)

#	FD	Iter. No	Total Training Time	Prep. Time	Loop Time
1	256	200	17.42 sec	2.39 sec	0.039 sec
2	256	5K	308.12 sec	2.42 sec	0.039 sec
3	256	10K	641.61 sec	2.40 sec	0.038 sec
4	256	20K	1318.37 sec	2.36 sec	0.039 sec
5	128	200	11.53 sec	2.39 sec	0.031 sec
6	128	5K	174.65 sec	2.37 sec	0.030 sec
7	128	10K	346.85 sec	2.56 sec	0.030 sec
8	128	20K	744.34 sec	2.38 sec	0.030 sec
9	64	200	7.91 sec	2.39 sec	0.026 sec
10	64	5K	110.00 sec	2.36 sec	0.026 sec
11	64	10K	215.97 sec	2.38 sec	0.026 sec
12	64	20K	447.20 sec	2.42 sec	0.026 sec

Total training time obviously is taken during training session, and expresses complete session time. The second and third values are taken during run time for data set test, in average. In preparation time, system allocates memory and restores models before starting a test loop. This value seems to be constant for all cases. But, loop time which will directly be FPS in hardware implementation goes low with less feature depth, as expected. Because there much fewer convolutional operations for shallow networks.

Alternative models of #4 and #2, with batch size changed to 2, gives training time of 1351.43 seconds and 320.02 seconds, respectively. Other two values do not even change as much as training time.

A model trained by NEU data set, with 256 FD, 8 batch size for 20000 iterations has taken around 488 seconds in total. The image size affects critically, in the end.

Similarly, average loop time is as low as 0.015 sec for this data set.

All in all, preparation time does not alter significantly. Training time goes up with more iterations and more features to be examined. But, system would work faster in run time if depth of features were less.

4.5.3.2 Detection Results

In this section, we will examine segmentation results of mid-end algorithm depending on metrics and visuals for network models. We will start with loss during training time, showing how it decreases over iterations, Fig. 4.39. It is lowered drastically after 100 iterations almost. So, average loss does not change significantly after this point.

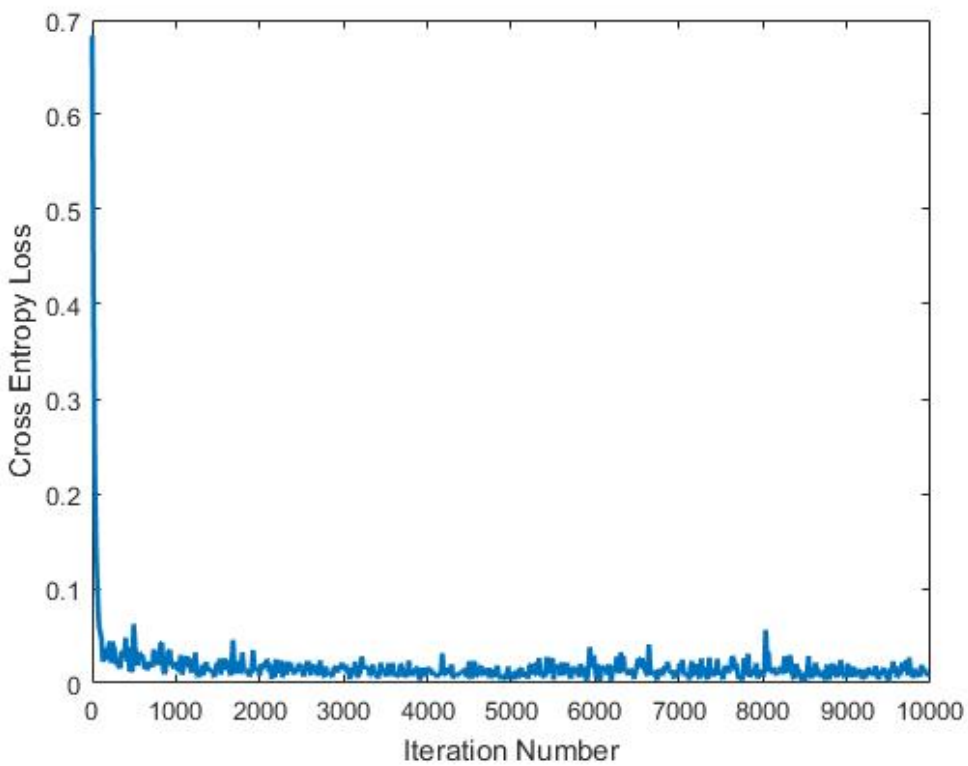


Figure 4.39: Decrease in loss versus iteration

Examining average loss values for test run in Hard set (Fig. 4.40), implies that loss decreases with deeper networks. Average loss also converges to very low values as number of iterations gets high.

Training Hard set with a 256 feature deep network for 20000 iterations, but with a batch size of 2, has given the best average loss, precisely 0.0124. The best we acquired with 8-batch models was 0.0125, that is still very successful. That result nominates the model (as *Model B*) to the next step, hardware implementation.

NEU data set modeling could go as low as 0.23 only. This is a high value compared Hard set. The reason behind may be image size difference and the mean area of an scratch object covers on image samples.

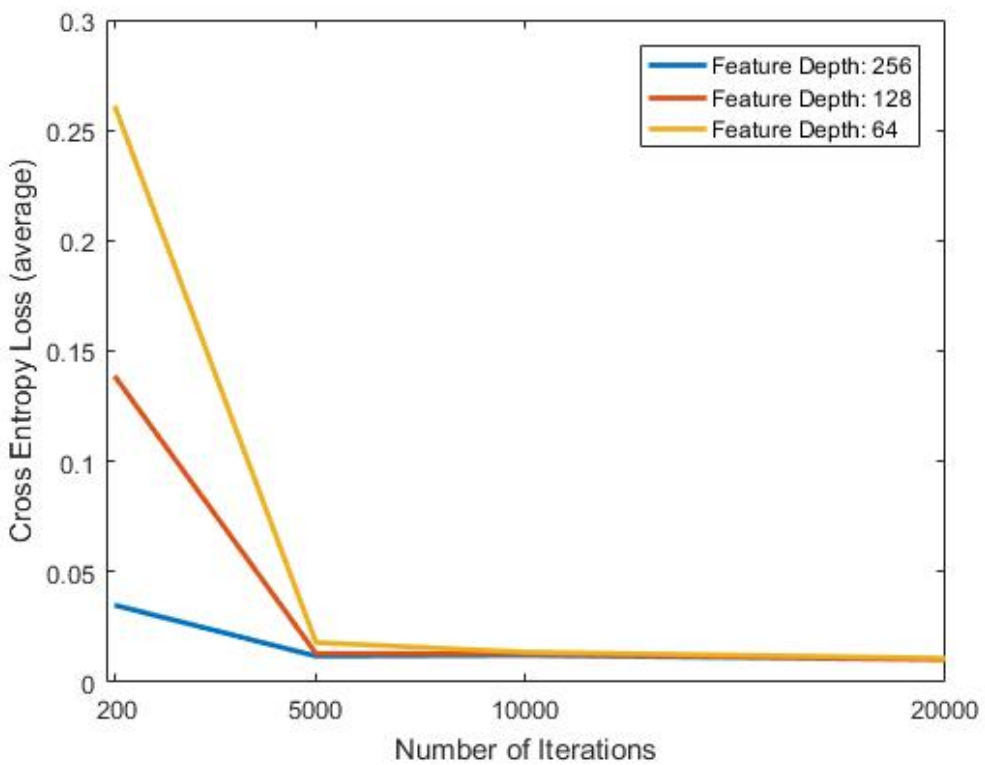


Figure 4.40: Loss versus number of iterations for different feature depths (Hard Set, batch size:8)

The loss results above demonstrates that feature depth is important on segmentation

performance. Moreover, batch size has a slight effect on the loss. But, this parameter also shows that 20000 iteration may not be that advantageous compared to 5000 iteration training. At this point, we should have a look at binary metrics.

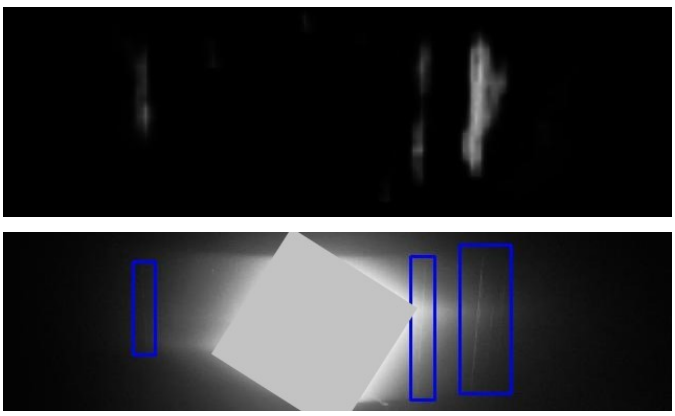
Table 4.11: Semantic metric results for deep neural network models (Hard Set, batch size:8)

#	FD	Iter. No	Accuracy(%)	Sensitivity(%)	Specificity(%)
1	256	200	64.0	97.1	63.9
2	256	5K	98.5	82.1	98.5
3	256	10K	98.8	90.8	98.8
4	256	20K	98.9	91.5	98.9
5	128	200	58.5	99.9	58.4
6	128	5K	99.0	81.3	99.0
7	128	10K	98.9	89.5	99.0
8	128	20K	99.1	90.6	99.1
9	64	200	58.5	99.9	58.4
10	64	5K	98.9	82.8	98.9
11	64	10K	98.9	88.2	98.9
12	64	20K	98.8	90.2	98.8

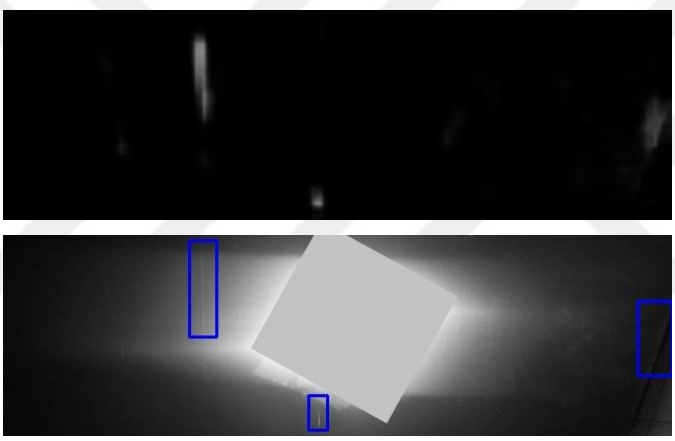
The binary metrics in Table 4.11, clearly display that some more iterations are needed after 5000 point. Please note that, there are shallow networks in the Table, with a high sensitivity or specificity. The proper examination in this evaluation should be two-sided. In other words, it is better to check if both metrics are acceptably high. For example, model number 9 may be misleading with a sensitivity of 99% and specificity of 58%.

Going over the binary metric results, model #4 draws attention with a successful segmentation. It is the second model (namely, *Model A*) which will be experimented on real-world applications. Model B still promises good results with an accuracy of 98.8% and sensitivity 91.5%.

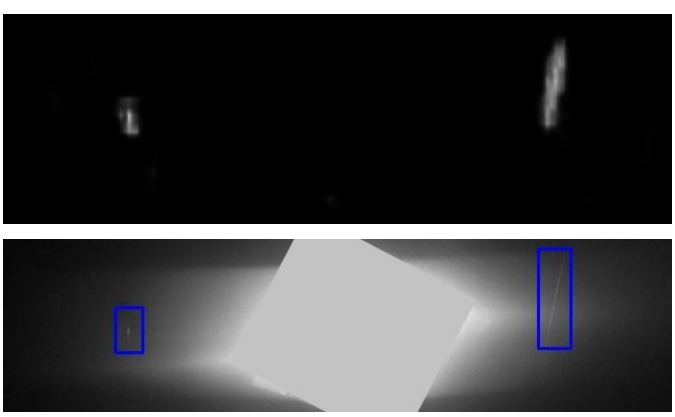
Here are some visual outputs of both nominated models in Figures 4.41, 4.42. Both are well-done on the task, but Model B is more aggressive and might need a higher *height filter*. The filters that turns soft results to rectangular boxes are explained in Section 4.5.3.3.



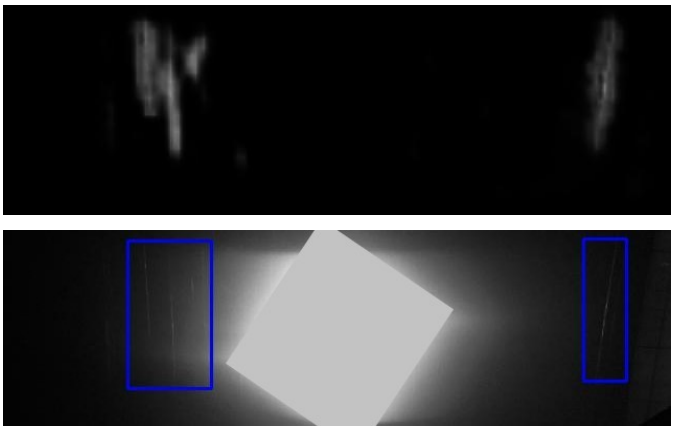
(a) Soft and Rectangular Result 1



(b) Soft and Rectangular Result 2

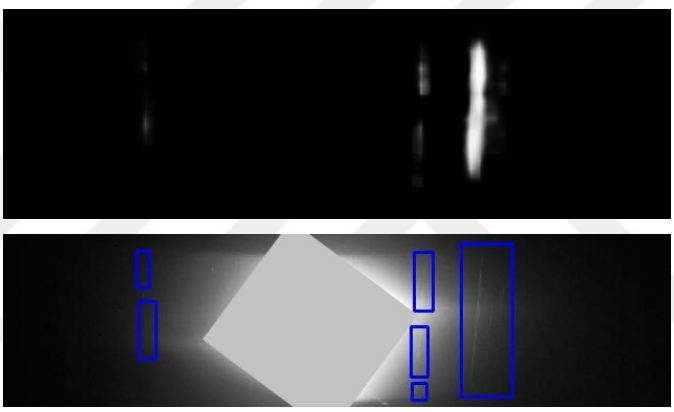


(c) Soft and Rectangular Result 3

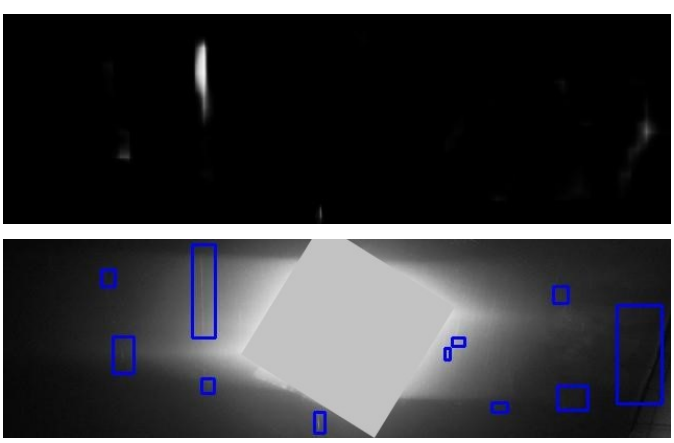


(d) Soft and Rectangular Result 4

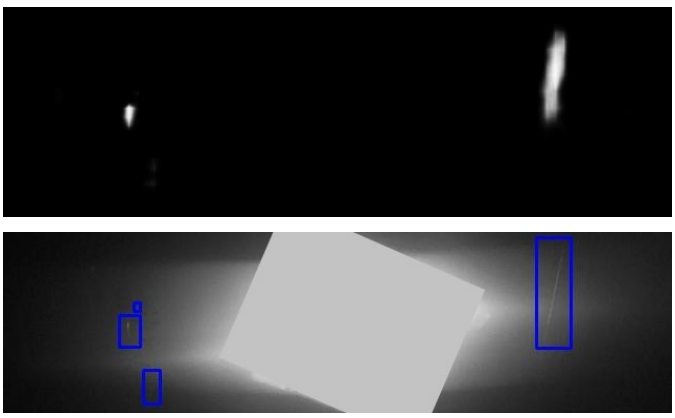
Figure 4.41: Resulting images for mid-end system modeling (Model A)



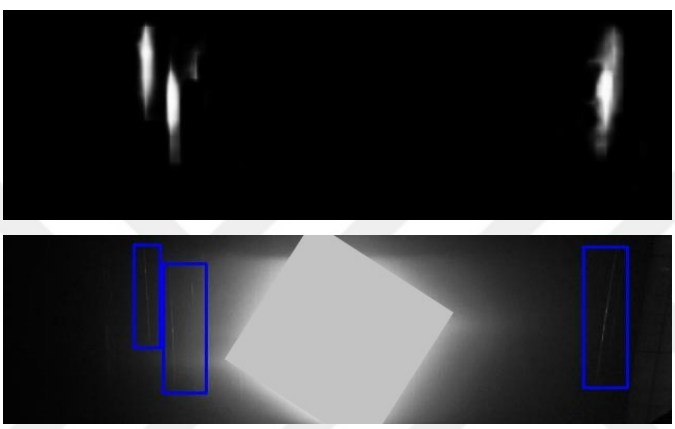
(a) Soft and Rectangular Result 1



(b) Soft and Rectangular Result 2



(c) Soft and Rectangular Result 3



(d) Soft and Rectangular Result 4

Figure 4.42: Resulting images for mid-end system modeling (Model B)

4.5.3.3 Evaluation

Regarding high timing performance (Sec. 4.5.3.1), precision in the dataset (Sec. 4.5.3.2) and robustness it serves, this model is qualified for a hardware implementation for the next stage.

The models differ in parameters like batch size, feature depth and number of iterations in training step. Some reasonable values are determined and evaluated with the help of data sets and segmentation metrics. Eventually, two of those models take attention, Table 4.12.

Table 4.12: Two chosen models for hardware experiments

	Strength	Feature Depth	Batch Size	# of Iterations
Model A	Best Binary Metrics	256	8	20000
Model B	Best Loss	256	2	20000

Both models shows sufficient timing performances for real-time application. In evaluation of such models, the file size of the whole parameters is important, actually. Because, the network designed would be transferred to a hardware in the end and this task may be impossible or inefficient for excessively large models. This is a problem for some models and coding studies are done for the sake of compression. The total file size goes up to 240MB for AlexNet, 552MB for VGG-16 [108].

For our models, only *feature depth* parameter affects the file size, as it is directly connected to the number of convolutional filters (Tab. 4.13).

Table 4.13: File Sizes of our networks

Feature Depth	Model File Size
256	4.9 MB
128	1.4 MB
64	523.5 kB

Finally, there are two more parameters of test-run. Those parameters has nothing to do with training, but can affect run time or test run results. They come in action when converting probabilistic soft results to discrete rectangular results, so they have a role in semantic segmentation, but for loss function. Those are namely, *soft filter* and *height filter*, fixed at values 10 and 25 respectively. Soft filter is the threshold for soft result to create a binary image, while height filter eliminates small rectangular

boxes around blobs in this binary.

Still, semantic metrics can be improved by tuning those filters. The table below (Tab. 4.14) proves that trained models still can be developed during run time applications.

Table 4.14: Fine-tuning of filters in run time on trained models

	Soft Filter	Height Filter	Sensitivity	Specificity
Model A	5	10	97.3	98.2
	5	25	97.2	98.2
Model B	5	10	96.4	98.6
	5	25	95.9	98.7

CHAPTER 5

CONCLUSION AND FUTURE WORK

This is the last part of the document apart from Appendices. There, final words on the work and a summary of all effort. Also, this part is supported with some idea to develop current system might be able to make it more efficient, more functional or less costly.

5.1 Conclusion

All in all, literature is reviewed enough and some techniques are picked among. Comparisons are made by modeling in controlled experiment, scientifically. And the result are presented for two different systems in ways as equal as possible.

Both systems may be preferred depending on the budget and environmental terms. Low-cost solution seems to be genuinely close to the lowest price level that such system could go. It is fairly successful, but prone to change in environmental factors. That can always be solved by fine tuning and better equipment, though.

Mid-end solution, otherwise, is completely robust and still in the price grades of acceptable as *inexpensive*. The training time is really short compared to popular networks and method is appropriate for supervised learning to be supported by workers.

In fact, a correct image acquisition was a main objective at first. Because, datasets, which is the essence of neural network training, are formed by those images. Also, the idea was that even a perfect linear object detection algorithm would not solve the problem, if the images acquired was under a certain quality and other measures. So, a considerable amount of time was taken by hardware studies and image forming by

cameras.

Another observation was that it is always easier to model on a software platform, but a hardware system to be finalized has its own hardships and disadvantages. In such works, as ours, software and hardware grow together, and even might make changes in each other. This last sentence summarizes the most difficult aspect of this and similar studies. Such studies form a bridge between distant visions of academics and industry, or theory and practice.

In short, we tried follow a scientific path by working over related work, examining potential hardware and making software models in advance. In the end, both methods proposed fulfilled app. more than 90% accuracy and can be used for different task of the same problem in real-time inspection. Please, see next section for further improvements.

5.2 Future Work

This section includes some idea on future possible work. Some of those directed to improve current methods, while some other may bring a whole new idea and require substantial amount of work.

Illumination techniques can always be developed. For example, there was an idea of light matrix in the beginning, so that system could span whole metal sheet. A primitive drawing is included in Figure 5.1. For switching, the circuit given in the end of Section 4.1.

Besides, other illumination configuration 2.4 can be tried. Especially, a polarizer other that a cheap linear one 2.9, for example, circular filter may be helpful. A ring light would possibly fail as it will create a large bright area in the middle, on the other hand, a structured light configuration can always improve emphasize on scratches.

Low-cost method does not consider the colors, in fact. It only focuses on the gradient of the original raw gray image. But, it can make use of the color of a scratch

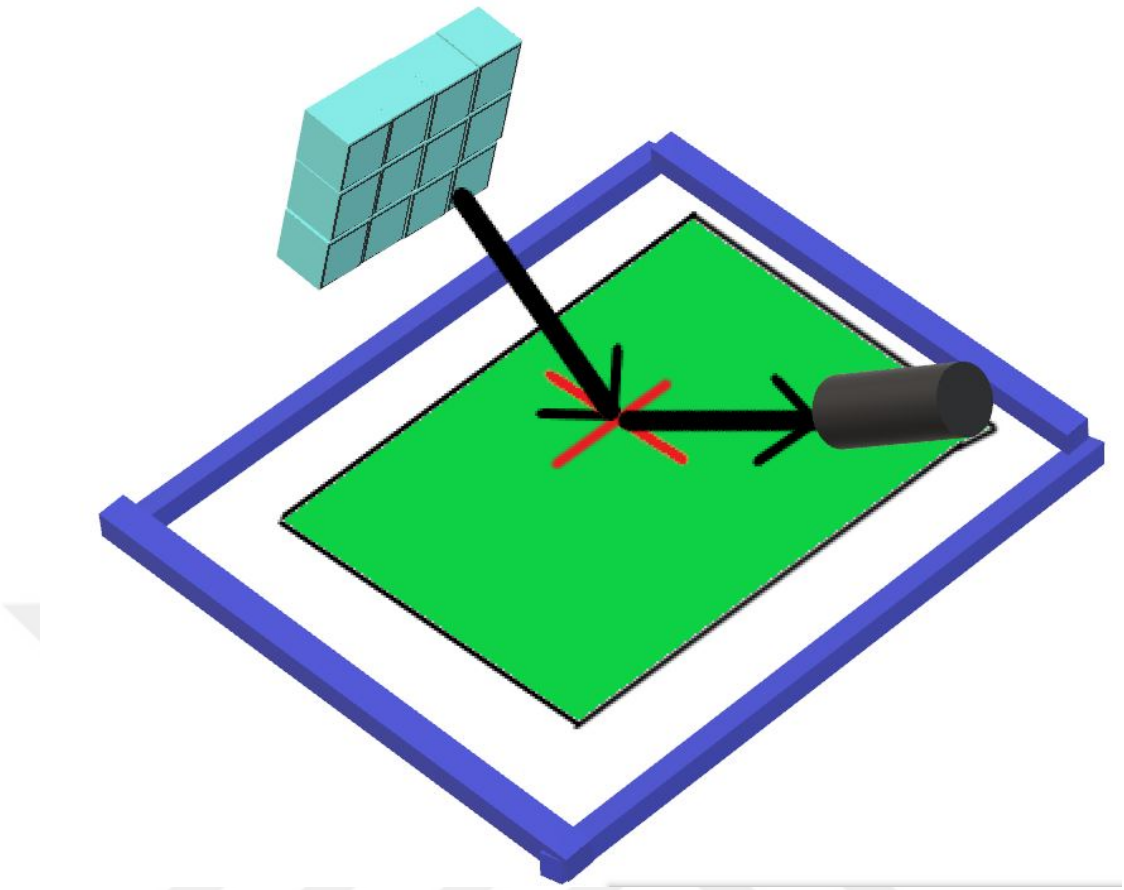


Figure 5.1: Primitive drawing of matrix of lights idea (numbers may vary)

which is white in the center and gets darker gray around (e.g. Generalized Hough Transform). In fact, some other aspects like randomness, line thickness checks and other application improvements can be done on Standard HT, with the help of this amazing study [91].

The CNN based model may be implemented on the Raspberry Pi 3. As seen in the Graph 3.4, Raspberry can handle small popular networks. But a smaller CNN model can be implemented (less than 9 convolutional layers) to get FPS greater than 1-2. Additionally, we may take advantage of a neural network USB stick, e.g. Movidius [109].

As this method needs a fine tuning under different environment circumstances, we

could consider generating a learning model with the Hough Transform based algorithm. As a result, with supervised learning, the parameters would be fine tuned [110] [111] [112].

In addition, Raspberry Pi performance can be enhanced by software overclocking [113]. Also, Python is a language for easier and neat research-development, but C++ is a better choice when it comes to final product. With these improvements a performance boost of 30% can be expected.

Mid-end method is also entirely open to improvement. In fact, this can be enough workload for even a new thesis. The proposed neural network may include some more or less layers of convolutions. Those convolutions may be separated into 3×1 and 1×3 instead of a single 3×1. In the separated case, there will be 6 parameters instead of 9 parameters as in the current version. Eventually, this will decrease the model size and operation time [114].

We could have gone for a more complex segmentation network like U-NET [115], which makes upsampling by deconvolution using information from earlier convolutions. Otherwise, there we may add a *dilated convolution* layer. This type adds zeros between numbers of 3×3 convolution kernels. In this way, with the same number of operations, o larger area can be analyzed by a single convolution layer [116].

A definite update would be using a professionally generated data set. A fixed distance for the final product should be determined before. After image acquisition, with a professional surface inspector, ground truth would be generated. This data set will result in better loss and accuracy performance, for sure.

Finally, we might perform a broader parameter adjustment. For example, we have not changed *learning rate* at all, from its default 0.001. Furthermore, *heigth filter* and *soft filter* could be changed in a more selection. Those parameters experimentally were decided to take their fixed values. They only affect sensitivity and specificity, though.

Other surface defects may be added. However, this need a lot more time and effort, as each type of defect needs a special study. But, for deep learning methods

robustness serves. Same networks may be successful without extra work.

REFERENCES

[1] B. Benchoff, "Introducing the raspberry pi 3." https://hackaday.com/2016/02/28/introducing-the-raspberry-pi-3/, Feb 2016.

[2] R. Foundation, "Raspberry pi 3 model b official page." https://www.raspberrypi.org/products/raspberry-pi-3-model-b/, 2018.

[3] Nvidia, "Nvidia jetson official page: The embedded platform for autonomous everything." https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/, 2017.

[4] JetsonHacks, "Nvidia jetson tx2 development kit." https://www.jetsonhacks.com/2017/03/14/nvidia-jetson-tx2-development-kit/, Mar 2017.

[5] D. Jones, "Camera hardware specs of raspberry pi camera module." https://picamera.readthedocs.io/en/latest/fov.html, 2017.

[6] P. Dunbar, *Machine Vision*. Byte, Jan 1986.

[7] R. J. Sanderson, *Machine Vision Systems: A Summary and Forecast*. Tech Tran Consultants Inc., 2nd ed., 1985.

[8] E. Optics, "Choose the correct illumination." https://www.edmundoptics.com/resources/application-notes/illumination/choose-the-correct-illumination/, 2018.

[9] "Reflection (physics)." https://www.wikiwand.com/en/Reflection_(physics), Accessed Aug 2018.

[10] "Dynamic range wikipedia page." https://www.wikiwand.com/en/Dynamic_range, Sep 2017.

[11] "Sharpness, acutance and resolution." http://www.photoreview.com.au/tips/shooting/sharpness,-acutance-and-resolution/, 2015.

[12] E. Benazera and J. Saksrisuwan, "Deepdetect performance report." https://github.com/jolibrain/dd_performances#deepdetect-performance-report, Sep 2017.

[13] CNXSoft, "Raspberry pi 3 board is powered by broadcom bcm2837 cortex a53 processor, sells for $35." https://www.cnx-software.com/2016/02/29/raspberry-pi-3-board-is-powered-by-broadcom-bcm2827-cortex-a53-processor-sells-for-35/, Feb 2016.

[14] "Jetson tx2 image." https://elinux.org/Jetson_TX2.

[15] N. Corp., "The raspberry pi camera module v2 official." https://www.raspberrypi.org/products/camera-module-v2/, 2016.

[16] "Cross-validation (statistics)." https://www.wikiwand.com/en/Cross-validation_(statistics), Accessed Sep 2018.

[17] "Hough transform wikipedia page." https://www.wikiwand.com/en/Hough_transform, Feb 2015.

[18] "Opencv 3.1.0 documentation: Canny edge detection." https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html, Accessed Aug 2018.

[19] "Mathematical morphology." https://www.wikiwand.com/en/Mathematical_morphology, Accessed Aug 2018.

[20] "Histogram equalization." https://www.wikiwand.com/en/Histogram_equalization, Accessed Aug 2018.

[21] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman, and K. Zuiderveld, "Adaptive histogram equalization and its variations," *Computer Vision, Graphics, and Image Processing*, vol. 39, no. 3, pp. 355 – 368, 1987.

[22] F. Strugar, "An investigation of fast real-time gpu-based image blur algorithms." https://software.intel.com/en-us/blogs/2014/07/15/an-investigation-of-fast-real-time-gpu-based-image-blur-algorithms, Accessed Aug 2018.

[23] F. Klocke and A. Kuchle, *Manufacturing Processes 1: Cutting*. RWTHedition, Springer Berlin Heidelberg, 2011.

[24] Cognex, "2d machine vision systems," *Cognex*, 2016.

[25] E. Butzer, "Camera-based, automatic surface inspections for accurate quality control during production," *Vitronic Co.*, 2018.

[26] Ametek, "Online detection, classification and visualization of defects," *Ametek*, 2016.

[27] W. Co., "Surface inspection, precise defect classification web ranger - surface inspection system," *Wintriss Co.*, 2017.

[28] D. F. Tech, "Film inspection, dynamic evaluation, system testing and development," *DarkField*, 2016.

[29] D. Industry, "Surface inspection system." http://www.directindustry.com/industrial-manufacturer/surface-inspection-system-103518.html.

[30] Alibaba.com, "Visual inspection systems prices." https://www.alibaba.com/showroom/visual-inspection-machines.html, Accessed Sep 2018.

[31] F. C. Campbell, ed., *Inspection of Metals—Understanding the Basics*. ASM International, 2013.

[32] T. Puntous, S. Pavan, D. Delafosse, M. Jourlin, and J. Rech, "Ability of quality controllers to detect standard scratches on polished surfaces," *Precision Engineering*, vol. 37, no. 4, pp. 924 – 928, 2013.

[33] C. Peterson, "How it works: The charged-coupled device, or ccd," *Journal of Young Investigators*, vol. 3, 2001.

[34] J. D. Meyer, “Nondestructive evaluation and quality control,” *ASM Handbook*, vol. 17, pp. 29–45, 1992.

[35] S. K. Sinha and P. W. Fieguth, “Automated detection of cracks in buried concrete pipe images,” *Automation in Construction*, vol. 15, no. 1, pp. 58 – 72, 2006.

[36] S.-N. Yu, J.-H. Jang, and C.-S. Han, “Auto inspection system using a mobile robot for detecting concrete cracks in a tunnel,” *Automation in Construction*, vol. 16, no. 3, pp. 255 – 261, 2007.

[37] F. Zhuang, Z. Yanzheng, L. Yang, C. Qixin, C. Mingbo, Z. Jun, and J. Lee, “Solar cell crack inspection by image processing,” in *Proceedings of 2004 International Conference on the Business of Electronic Product Reliability and Liability (IEEE Cat. No.04EX809)*, pp. 77–80, April 2004.

[38] F. C. Mayrhofer and K. Niel, “Optical crack detection of refractory bricks,” *ECDNT*, no. 70, 2006.

[39] K. A. Gross, J. Lungevics, J. Zavickis, and L. Pluduma, “A comparison of quality control methods for scratch detection on polished metal surfaces,” *Measurement*, vol. 117, pp. 397 – 402, 2018.

[40] Y. M. W. R. H. B. D. B. Verma, “Chemical mechanical polishing compositions for metal and associated materials and method of using same,” *Advanced Technology Materials Inc.*, 2001-08-14.

[41] C. T. Wang, N. Gao, M. G. Gee, R. J. Wood, and T. G. Langdon, “Effect of grain size on the micro-tribological behavior of pure titanium processed by high-pressure torsion,” *Wear*, vol. 280-281, pp. 28 – 35, 2012.

[42] E. Akman and E. Cerkezoglu, “Compositional and micro-scratch analyses of laser induced colored surface of titanium,” *Optics and Lasers in Engineering*, vol. 84, pp. 37 – 43, 2016.

[43] J. Arias, M. Mayor, J. Pou, Y. Leng, B. León, and M.Pérez-Amor, “Micro- and nano-testing of calcium phosphate coatings produced by pulsed laser deposition,” *Biomaterials*, vol. 24, no. 20, pp. 3403 – 3408, 2003.

[44] D. Rats, V. Hajek, and L. Martinu, "Micro-scratch analysis and mechanical properties of plasma-deposited silicon-based coatings on polymer substrates," *Thin Solid Films*, vol. 340, no. 1, pp. 33 – 39, 1999.

[45] L. Zhang and R. Koch, "An efficient and robust line segment matching approach based on lbd descriptor and pairwise geometric consistency," *Journal of Visual Communication and Image Representation*, vol. 24, no. 7, pp. 794 – 805, 2013.

[46] J. C. C. Tikhe, "Metal surface inspection for defect detection and classification using gabor filter," *Int J Innov Res Sci Eng Tech*, 2014.

[47] A. O. M. Luiz, L. C. P. Flávio, and E. M. A. Paulo, "Automatic detection of surface defects on rolled steel using computer vision and artificial neural networks," in *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, pp. 1081–1086, Nov 2010.

[48] T. Wang, Y. Chen, M. Qiao, and H. Snoussi, "A fast and robust convolutional neural network-based defect detection model in product quality control," *The International Journal of Advanced Manufacturing Technology*, vol. 94, pp. 3465–3471, Feb 2018.

[49] A. Chondronasios, I. Popov, and I. Jordanov, "Feature selection for surface defect classification of extruded aluminum profiles," *The International Journal of Advanced Manufacturing Technology*, vol. 83, pp. 33–41, Mar 2016.

[50] H. W. Lippincott and H. Stark, "Optical–digital detection of dents and scratches on specular metal surfaces," *Appl. Opt.*, vol. 21, pp. 2875–2881, Aug 1982.

[51] R. Seulin, F. Mérienne, and P. Gorria, "Machine vision system for specular surface inspection : Use of simulation process as a tool for design and optimization," 2002.

[52] E. Hecht and A. Zajac, *Optics*. Addison-Wesley world student series, Addison-Wesley, 1987.

[53] I. 12232:2006, "Photography – digital still cameras – determination of exposure index, iso speed ratings, standard output sensitivity, and recommended exposure index," *ISO*, vol. 2, 2006.

[54] "Color balance wikipedia page." https://www.wikiwand.com/en/Color_balance, Aug 2016.

[55] J. A. S. Viggiano, "Comparison of the accuracy of different white-balancing options as quantified by their color constancy," 2004.

[56] E. Peli, "Contrast in complex images," *J. Opt. Soc. Am. A*, vol. 7, pp. 2032–2040, Oct 1990.

[57] B. Peterson, *Understanding Exposure, Fourth Edition: How to Shoot Great Photographs with Any Camera*. Potter/Ten Speed/Harmony/Rodale, 2016.

[58] G. Lawrance, "Exposure compensation." https://www.geofflawrence.com/exposure_compensation.html, 2016.

[59] S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High dynamic range video," *ACM Trans. Graph.*, vol. 22, pp. 319–325, July 2003.

[60] B. Yusuf, "Best single board computers 2018 (raspberry pi alternatives)." "https://all3dp.com/1/single-board-computer-raspberry-pi-alternative/", Apr 2018.

[61] "What are the best single-board computers?." "https://www.slant.co/topics/1629/~single-board-computers", 2018.

[62] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *CoRR*, vol. abs/1609.07061, 2016.

[63] A. Sachan, "Embedded computer vision: Which device should you choose?." "https://www.learnopencv.com/embedded-computer-vision-which-device-should-you-choose/", Feb 2017.

[64] "Raspberry pi wikipedia page." https://www.wikiwand.com/en/Raspberry_Pi#, Mar 2018.

[65] M. Harris, "Inside pascal: Nvidia's newest computing platform," *Nvidia Developer Blog*, Apr 2016.

[66] P. Lawrance, "Meet jetson, the platform for ai at the edge," *Nvidia Developer Blog*, 2017.

[67] Sony, "Imx219pq: Diagonal 4.6 mm (type 1/4.0) 8.08m-effective pixel color cmos image sensor." https://www.sony-semicon.co.jp/products_en/new_pro/april_2014/imx219_e.html, 2016.

[68] K. Doris, "To ccd or to cmos, that is the question." https://www.bhphotovideo.com/find/newsLetter/Comparing-Image-Sensors.jsp, 2017.

[69] "Ov5693: Color cmos 5-megapixel (2592x1944) image sensor with omnibsi-2™ technology." https://www.ovt.com/sensors/OV5693, Accessed Aug 2018.

[70] P. S. Toh, "Line scan camera patent." https://patents.google.com/patent/US6292608B1/en, 1999-09-30.

[71] E. Tutorials, "Relay switch circuit." https://www.electronics-tutorials.ws/blog/relay-switch-circuit.html, 2014.

[72] K. Song and Y. Yan, "Neu surface defect database." http://faculty.neu.edu.cn/yunhyan/NEU_surface_defect_database.html, 2013.

[73] K. Song and Y. Yan, "A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects," *Applied Surface Science*, vol. 285, pp. 858 – 864, 2013.

[74] E. Fernandez-Moral, R. Martins, D. Wolf, and P. Rives, "A new metric for evaluating semantic segmentation: leveraging global and contour accuracy," in *Workshop on Planning, Perception and Navigation for Intelligent Vehicles, PPNIV17*, (Vancouver, Canada), Sept. 2017.

[75] "Sensitivity and specificity." https://www.wikiwand.com/en/Sensitivity_and_specificity, Accessed Aug 2018.

[76] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of operations research*, vol. 134, no. 1, pp. 19–67, 2005.

[77] G. McLachlan, K. Do, and C. Ambroise, *Analyzing Microarray Gene Expression Data*. Wiley Series in Probability and Statistics, Wiley, 2005.

[78] P. V. C. Hough, "Method and means for recognizing complex patterns." `https://patentimages.storage.googleapis.com/9f/9f/f3/87610ddec32390/US3069654.pdf`, 1960-03-25.

[79] M. Bôcher, *Plane Analytic Geometry: With Introductory Chapters on the Differential Calculus*. H. Holt, 1915.

[80] Y. Mochizuki, A. Torii, and A. Imiya, "N-point hough transform for line detection," *Journal of Visual Communication and Image Representation*, vol. 20, no. 4, pp. 242 – 253, 2009.

[81] D. Ballard, "Generalizing the hough transform to detect arbitrary shapes," *Pattern Recognition*, vol. 13, no. 2, pp. 111 – 122, 1981.

[82] H. Yuen, J. Princen, J. Illingworth, and J. Kittler, "Comparative study of hough transform methods for circle finding," *Image and Vision Computing*, vol. 8, no. 1, pp. 71 – 77, 1990.

[83] L. Xu, E. Oja, and P. Kultanen, "A new curve detection method: Randomized hough transform (rht)," *Pattern Recognition Letters*, vol. 11, no. 5, pp. 331 – 338, 1990.

[84] Z. Yao and W. Yi, "Curvature aided hough transform for circle detection," *Expert Systems with Applications*, vol. 51, pp. 26 – 33, 2016.

[85] N. Kiryati, Y. Eldar, and A. Bruckstein, "A probabilistic hough transform," *Pattern Recognition*, vol. 24, no. 4, pp. 303 – 316, 1991.

[86] J. Illingworth and J. Kittler, "The adaptive hough transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, pp. 690–698, Sept 1987.

[87] H. Li, M. A. Lavin, and R. J. L. Master, "Fast hough transform: A hierarchical approach," *Computer Vision, Graphics, and Image Processing*, vol. 36, no. 2, pp. 139 – 161, 1986.

[88] S. Guo, T. Pridmore, Y. Kong, and X. Zhang, "An improved hough transform voting scheme utilizing surround suppression," *Pattern Recognition Letters*, vol. 30, no. 13, pp. 1241 – 1252, 2009.

[89] N. Pozzobon, F. Montecassiano, and P. Zotto, "A novel approach to hough transform for implementation in fast triggers," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 834, pp. 81 – 97, 2016.

[90] V. Leavers, "Which hough transform?," *CVGIP: Image Understanding*, vol. 58, no. 2, pp. 250 – 264, 1993.

[91] P. Mukhopadhyay and B. B. Chaudhuri, "A survey of hough transform," *Pattern Recognition*, vol. 48, no. 3, pp. 993 – 1010, 2015.

[92] A. M. Reza, "Realization of the contrast limited adaptive histogram equalization (clahe) for real-time image enhancement," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 38, pp. 35–44, Aug 2004.

[93] L. MacDonald, *Digital Heritage*. Taylor & Francis, 2006.

[94] H. Andrews and B. Hunt, *Digital image restoration*. Prentice-Hall signal processing series, Prentice-Hall, 1977.

[95] W. McCulloch and W. Pitts, "logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. vol. 5, pp. 115–133, 1943.

[96] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[97] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, pp. 400–407, 09 1951.

[98] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, July 2011.

[99] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[100] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[101] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 1980.

[102] I. S. A. Krizhevsky and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 NIPS'12*, pp. 1097–1105, 2012.

[103] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[104] "Gstreamer pipelines for tegra x2." `https://developer.ridgerun.com/wiki/index.php?title=Gstreamer_pipelines_for_Tegra_X2#nvcamerasrc_2`, Accessed at 2018.

[105] A. Ouaknine, "Review of deep learning algorithms for object detection." `https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852`, Accessed at 2018.

[106] "An open source machine learning framework for everyone." `https://www.tensorflow.org/`, Accessed Aug 2018.

[107] "Deep learning software." `https://developer.nvidia.com/deep-learning-software`, Accessed Aug 2018.

[108] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[109] "Powering the machine intelligence revolution." https://www.movidius.com/, Accessed Aug 2018.

[110] E. S. McVey, G. L. Dempsey, J. Gavcia, and R. M. Inigo, "Artificial neural network implementation of the hough transform," in *Twenty-Third Asilomar Conference on Signals, Systems and Computers, 1989.*, vol. 1, pp. 128–132, Oct 1989.

[111] C. K. Chan and M. B. Sandler, "A neural network shape recognition system with hough transform input feature space," in *1992 International Conference on Image Processing and its Applications*, pp. 197–200, April 1992.

[112] J. Basak and S. K. Pal, "Hough transform network," *Electronics Letters*, vol. 35, pp. 577–578, April 1999.

[113] A. Rosebrock, "Optimizing opencv on the raspberry pi." https://www.pyimagesearch.com/2017/10/09/optimizing-opencv-on-the-raspberry-pi/, Oct 2017.

[114] L. Kaiser, A. N. Gomez, and F. Chollet, "Depthwise separable convolutions for neural machine translation," *arXiv*, 2017.

[115] O. R. P. Fischer and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015.

[116] P. Moeskopsa, M. Veta, M. W. Lafarge, K. A. J. Eppenhof, and J. P. W. Pluim, "Adversarial training and dilated convolutions for brain mri segmentation," *CoRR*, vol. abs/1707.03195, 2017.

[117] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, pp. 679–698, Nov 1986.

[118] "Sobel edge detector." http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm, Accessed Aug 2018.

[119] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, pp. 62–66, Jan 1979.

[120] J. Serra, *Image Analysis and Mathematical Morphology*. Orlando, FL, USA: Academic Press, Inc., 1983.

[121] J. C. Russ, *Image Processing Handbook, Fourth Edition*. Boca Raton, FL, USA: CRC Press, Inc., 4th ed., 2002.

[122] L. Shapiro and G. Stockman, *Computer Vision*. Prentice Hall, 2001.

[123] "Gaussian filtering wikipedia page." `https://www.wikiwand.com/en/Gaussian_filter`, Accessed Aug 2018.

[124] "Opencv 3.1.0 documentation: Image processing filtering." `https://docs.opencv.org/3.1.0/d4/d86/group__imgproc__filter.html#gac05a120c1ae92a6060dd0db190a61afa`, Accessed Aug 2018.

APPENDIX A

AUXILIARY TOOLS

A.1 Some Methods Commonly Used in Image Processing

In the following section, a *short explanation* for each method is given concerning image processing applications. Note that, those methods may or may not be implemented in our project. Some of them were implemented at some point, then decided to be inefficient or unnecessary. Some of them are only explained shortly due to their place in the literature.

A.1.1 Canny Edge Detector

This is method for detection of edge features in an 2-D array, alternatively an image. The original algorithm is developed by J. Canny in 1986 [117]. The method is formed by a multi-stage approach.

Gaussian filtering is the first step in the original paper to get rid of the noise related false edges. This blurring technique may or may not be included in Canny Edge Detector functions of different software languages. In our case, this step separately examined (App. A.1.5) and also coded before Canny filter.

Intensity gradient of the image is , in fact, the major feature data to be analyzed by most of the edge filtering algorithms. There many matrix filters to unveil gradient features of an image: Sobel, Prewitt, Roberts. The kernel size may be user-defined, but it conventional to use 3×3 kernels, as they the smallest possible in odd number. A

Sobel filtered image consists of two feature extractor in horizontal (G_x) and vertical (G_y), see Eqn. A.1 [118].

$$A : inputimage$$

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

$$G_{sobel} = \sqrt{G_x^2 + G_y^2} \tag{A.1}$$

Non-maximum suppression is an edge-thinning technique used, because a good edge detector should return center point of a thick edge. Figure A.1 illustrates this method: every pixel is checked if it is a local maximum in the gradient direction. Edge, in the example, is vertical. Point A is checked if it is a maximum in relation with B-C points, which are points on the gradient direction. If so, Point A stays as an edge pixel, otherwise suppressed.

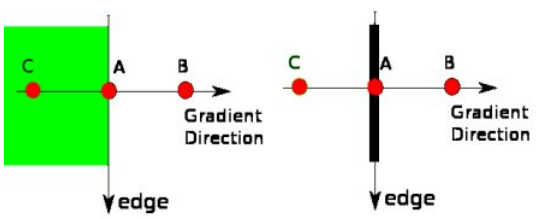


Figure A.1: Non-maximum Suppression [18]

Double thresholding is one of the novelties of Canny Edge Detector. This step is also called *Hysteresis Search*, due to tracking mechanism with two different thresholds. The remaining pixels with gradient intensity higher than $maxVal$ (Fig. A.2) are considered *strong edge pixels*. Anything lower that $minVal$ are discarded in the same

fashion. The values between are checked if they are connected to a *strong edge pixel* with 8- or 4- connectivity (user defined). If so, they are picked in the resulting edge map, as well. After double thresholding step, Canny Edge Filtered image is acquired.

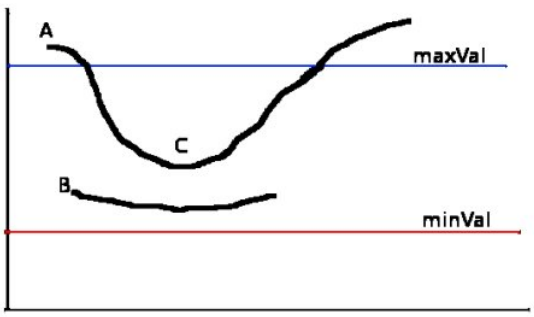


Figure A.2: Double Hysteresis Thresholding [18]

A.1.2 Otsu's Threshold Method

This automatic thresholding method was used in early stages of coding. As it is highly related to our topic, it should be briefly mentioned in this context. This method could have used at mask thresholding step (fixed number in current version of the code) and also can help at Hough, Gaussian or CLAHE parameter choices.

Otsu method aims to divide given image into two classes (foreground, background) such that *intra-class variance* (Eqn. A.2) is minimized. This variance is calculated as weighted sum of variances of those two classes [119]:

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t) \tag{A.2}$$

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

L: number of histogram bins, t: Otsu's Threshold, $p(i)$: pixel count at bin i

A.1.3 Morphological Operations

In any image processing application, working with masks, binary or gray images, it is probable to encounter a morphological operation (also *Mathematical Morphology*). Those operations are in fact , sliding matrices with different kernel shapes (*structuring elements*), values and simple logical rules. Consequently, a new image of the same data type is obtained, with desired filtering applied. This filtering operation may be removing holes, eliminating certain shapes pixel clusters, detection certain blobs, extracting granular shapes and so on. The theory is supported by many fields of math, mainly topology and geometrical continuous-space concepts like convexity, connectivity, geodesy. On the other hand, what we are going to include shortly are the most simple operation: dilation, erosion , opening and closing. However, there are some other popular morphological transforms in image processing literature [120]:

- Hit-or-miss Transform
- Pruning Transform
- Skeletonization (medial axis trans.)
- Thickening, Thinning
- Watershed Transform
- Reconstruction
- Granulometry
- Geodesic Distance
- Top-hat, Bottom-hat Transforms
- etc.

Dilation is one of two most basic operations in this context. A user-defined structuring element (B) is windowed around the image, and for every foreground pixel of

input image (A), structuring element is superimposed. This operation is shown with following equation (Eqn. A.3) (Fig. A.3a):

$$A \oplus B = \bigcup_{b \in B} A_b = \{z \in E | \left(B^s\right)_z \cap A \neq \varnothing\} \quad where \tag{A.3}$$

$$B^s = \{x \in E | -x \in B\}$$

Erosion is the other basic operation, and the dual of dilation. A user-defined structuring element (B) is windowed inside foreground pixels of input image (A) this time. Only those pixels correspoding to center point of B remain, if whole structuring element fits in foreground area of A. Formally the operation is shown with following equation (Eqn. A.4) (Fig. A.3b):

$$A \ominus B = \bigcap_{b \in B} A_{-b} = \{z \in E | B_z \subseteq A\} \quad where \tag{A.4}$$

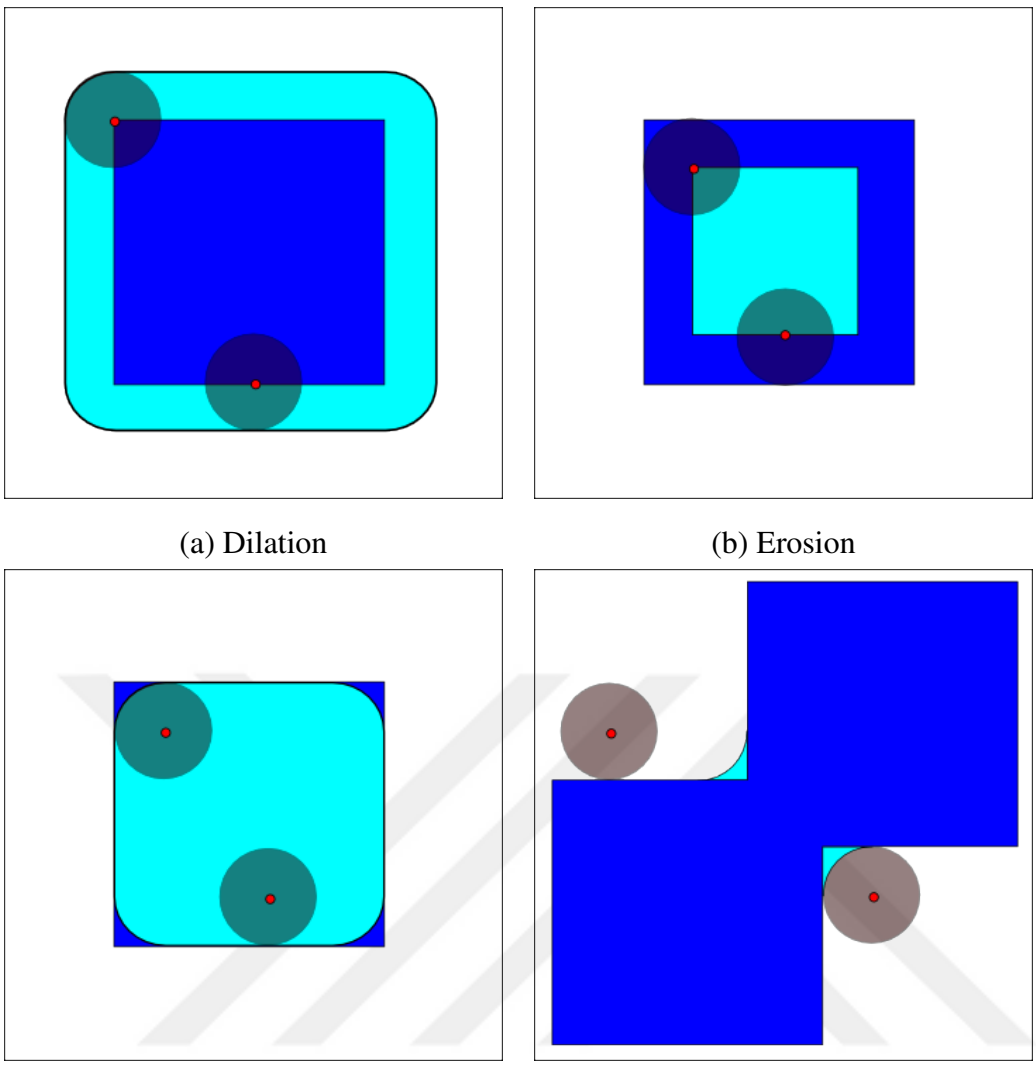
$$B_z = \{b + z | b \in B\}, \forall z \in E$$

Image opening is simply an erosion of A by B, followed by dilation by same structuring element (Eqn. A.5). This basically removes shapes smaller than B element, but does not distort other details in A (Fig. A.3c).

$$A \circ B = (A \ominus B) \oplus B \tag{A.5}$$

Image closing is simply an dilation of A by B, followed by erosion by same structuring element (Eqn. A.6). This operation closes the holes and makes connections between foreground of A depending on element B (Fig. A.3d).

$$A \bullet B = (A \oplus B) \ominus B \tag{A.6}$$



(a) Dilation (b) Erosion

(c) Opening (d) Closing

Figure A.3: Basic morphological operations in image processing [19]

A.1.4 Histogram Equalization

This is another image processing tool, of contrast adjustment. The algorithm is based on the histogram of the image, as the name suggests. Histogram of an image is count of pixels divided into intensity bins that has the same gray value. If an image has a histogram distribution that is concentrated in a small region, then it is low contrast. To make details visible without changing total brightness, histogram equalization is used as a common method. By memorizing a look-up table, the operation can be reversed

easily.

To understand main transform, we should state what a CDF is. *Cumulative Distribution Function* is the sum of probabilities for given index and indexes before. So, starting from the first bin it accumulates the pixel probabilities (Eqn. A.7). Here, $p_x(i)$ denotes the probability of a pixel to be found in a histogram bin i of histogram of image x. The number bins L is typically 256 [121].

$$cdf_x(i) = \sum_{j=0}^{i} p_x(j) \quad where \tag{A.7}$$

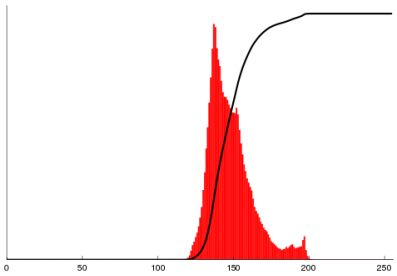
$$p_x(i) = \frac{n_i}{n}, \ \ 0 \leqslant i < L$$

Using knowledge of CDF, the aim is to acquire a single slope line on CDF graph (Fig. A.4d). Because, this is equivalent to having a uniform histogram, in which the contrast distribution is perfect. A general formula for the method is given in Eqn. A.8. An example in Figure A.4, clearly shows improvement in contrast.

$$h(v) = round\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right) \quad where \tag{A.8}$$

$M \times N$: *image dimensions,* cdf_{min} *: minimum value of CDF, L : number of bins*

(a) Before Histogram Equalization



(b) Initial CDF and histogram

(c) After Histogram Equalization

(d) Transformed CDF and histogram

Figure A.4: Histogram Equalization with images, CDFs (black lines) and histograms (red bins) [20]

The most significant downside of equalization using whole image histogram is that it also amplifies noise. Modifications on this method, hence, use multiple histograms (*subhistograms*) for local contrast dependence. A widely used variation is called *Contrast Limited Adaptive Histogram Equalization (CLAHE)*, in which a window is sliding across the image and applies histogram equalization in those tiles separately [21]. While doing so, histogram is clipped by a defined value and clipped pixels are distributed uniformly to the rest of the histogram, as in Figure A.5. Suggested parameters are 8×8 tile size and 3-4 clip limit (or contrast limit) [92].

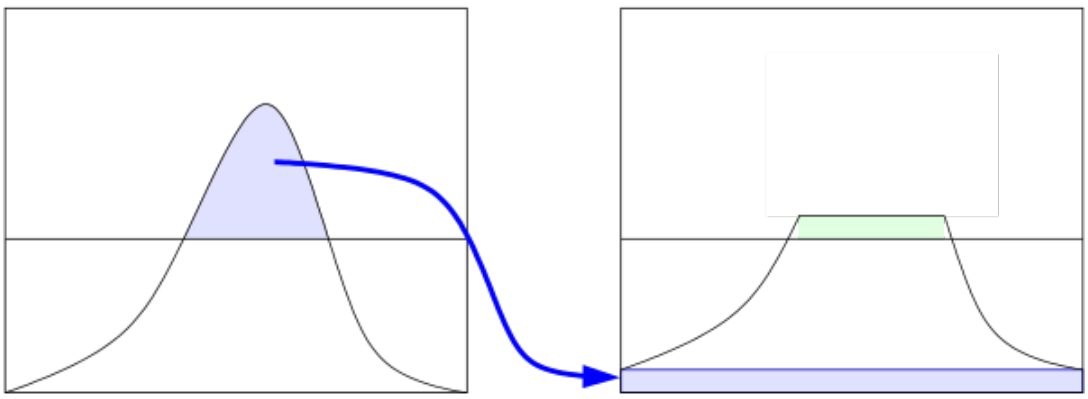


Figure A.5: CLAHE histogram clipping [21]

A.1.5 Gaussian Filtering

Gaussian filters are in low-pass characteristics, and used in many applications. But, here we will explain shortly its role in image filtering. The main purpose of a Gaussian filter applied on an image is blurring. This may be due to intended blur effect, however, in many cases its power to eliminate noise. To prevent false detection, the image is smoothed.

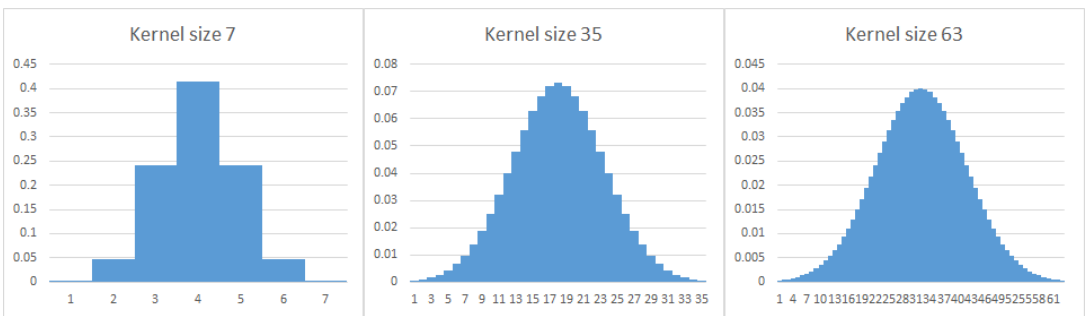


Figure A.6: Digital Gaussian Filter with different kernel sizes [22]

A Gaussian kernel is the matrix representation of this method, to be convolved with the image by sliding window procedure. Those kernels are digitized (quantized) versions of smooth Gaussian functions (Fig. A.6), and usually depicted in square form.

Below, an equation (A.9) for kernel with size $(2k+1) \times (2k+1)$ is given [122]:

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right);\ 1 \leq i, j \leq (2k+1) \quad \text{(A.9)}$$

Choosing a Gaussian kernel with the correct size for specific problem is important. A large Gaussian filter may reduce SNR in whole image, while edge details may be lost in the same proportion, so that edge features could be harder to find. Kernel with sizes 3×3 are 5×5 are commonly used [123].

Lastly, *sigma* σ parameter denotes the square root of standard deviation. The greater sigma would result in more blurred image. There is no single way to determine this parameter which is user-dependent. In software languages, it is usually default valued (commonly 0.5) or calculated by kernel size (see Eqn. A.10, for OpenCV [124]).

$$\sigma = 0.3 \times ((ksize - 1) \times 0.5 - 1) + 0.8 \quad \text{(A.10)}$$

A non-square shaped Gaussian Filter is found by matrix multiplication of two one-dimensional filters. For example, multiplying 7×1 and 1×3 kernel sized filters results in a 7×3 filter.

A.2 Hough Transform

Classical Hough Transform aimed to be able to detect imperfect lines in a simple way. Given a 2-D matrix (an image) with binary pixel values, either 1 or 0 each. Then for each pixel having the value 1 following line equation is solved in Hesse form

$$r = x\cos\theta + y\sin\theta [79]$$

Here r stands for the perpendicular distance of the line to the origin. Note that, every language and framework has its own origin definition, but it is usually located at upper left or lower left corner of the image. In a few cases, it located at the center. Furthermore, θ is the angle that x-axis makes with the perpendicular line from origin

to the line with given equation. Depiction of both parameters is given in the following Figure A.7.

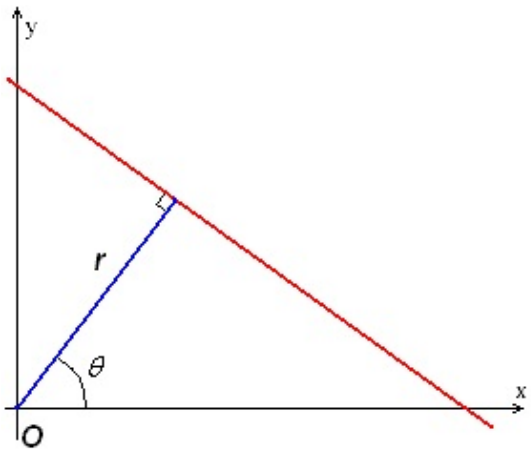


Figure A.7: Demonstration of line equation parameters [17]

Then, a 2-D matrix is generated for each r and θ value. The step size for each parameters are defined by user. It is usual to enter step of 1 pixel for r and 1 degree for θ. This 2-D matrix is called *Hough space*. For each pixel having value 1, the number in the cell corresponding (r, θ) pair in Hough space is incremented.

After, all image pixels with value 1 are visited, depending on the parameter *Hough threshold*, the line candidates in accumulator matrix are filtered. This filter may be directly thresholding whole matrix or may be local maxima search. The main transform ends here returning orientation and distance to origin information of lines [78]. Next step may be finding pixels and line location that matches with the strongest line candidates.

Here, is a toy example of an image having 2 different lines. Black pixels have value 1, while white pixels are 0. In Figure A.8b, resulting Hough space is given. Please notice that there are two points that accumulation has high values.

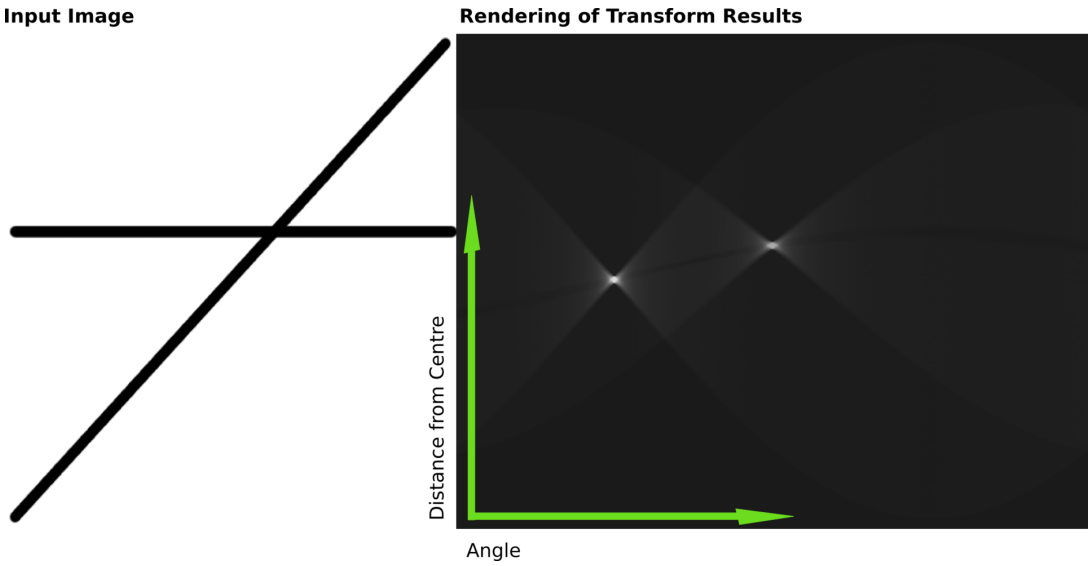


Figure A.8: A Hough Transform example applied to an image with 2 lines [17]

A.2.1 Variation of Hough Transform

In literature, Standard Hough Transform logic did not stop expanding until today. Although, there are still some improvements for line detection in this manner [80] , usually new Hough Transform covers other shape detection. Methods with the purpose of multi shape detection are called *Generalized Hough Transform* [81]. The most popular one is obviously circles [82] and ellipses, in other words curvatures. Those new methods often points to slowness and memory demands of original method. Some of them proposed solution based on randomization and RHT (Randomized Hough Transform) is coined [83]. In the same way, others make some pre-estimations (e.g. circle radius) to limit candidates [84].

Probabilistic Hough Transform has a significant importance as it is almost 30% faster in general [85]. The idea behind is elimination of some points randomly to lower the number of operations in a single image. But in OpenCV implementation, the results were not satisfying that we did not prefer to use this method.

Adaptive Hough Transform also called *Multi-Scale Hough Transform*, is another decent method. In that one, the way of how to find peaks in accumulator matrix, changes. There are 2 more parameters that come in action called *srn* and *stn*. Those parameters divide corresponding step sizes of r and θ, resulting numbers would be accurate accumulator resolution. Consequently, two level approach is implemented in Hough space [86]. This transform yields better result at finding weak lines, but operation time rockets up (from 0.15 seconds to 2 seconds) in OpenCV frame, so it was not preferable.

Fast Hough Transform in fact, divides and conquers the image given. It, recursively, aparts the image given into smaller rectangular area from low to high resolution, while calculating Hough Transform for each. If an area cannot exceed the threshold, then that area is skipped for further transforms. In short, number of candidates is limited and operation count is decreased [87].

One of recent articles made a new approach to simple accumulation function, and changed to a complex one that considers edge pixel with small weights. In this Canny (App. A.1.1) resembling approach, they take stronger edges for accumulation. This phenomenon of edge weight consideration is called *isotropic surrounding suppression*. Their objective is to reduce false positives in real-world images [88]. Another one made use of parallel process power of FPGAs [89].

Having difficulties on choosing the right Hough Transform, there are valuable studies on surveying HTs [90]. All in all, Hough Transform has proven itself to be a great tool, as there are more than 2500 published material concerning generalization, improvement, variant and application of Hough Transform [91]. After examining variations, and experimenting several, we have decided to continue with the original approach to keep it smart and simple.