

**SLOVAK UNIVERSITY OF TECHNOLOGY
IN BRATISLAVA**

FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

Reg. No.: FCHPT-16584-115961

Process Variable Monitoring Using Camera Data

MASTER THESIS

**SLOVAK UNIVERSITY OF TECHNOLOGY
IN BRATISLAVA**

FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

Reg. No.: FCHPT-16584-115961

Process Variable Monitoring Using Camera Data

MASTER THESIS

Study programme:	Automation and Information Engineering in Chemistry and Food Industry
Study field:	Cybernetics
Training workspace:	Institute of Information Engineering, Automation, and Mathematics
Thesis supervisor:	doc. Ing. Radoslav Paulen, PhD.



MASTER THESIS TOPIC

Student: **Bc. Jakub Gaborčík**
Student's ID: 115961
Study programme: Automation and Information Engineering in Chemistry and Food Industry
Study field: Cybernetics
Thesis supervisor: doc. Ing. Radoslav Paulen, PhD.
Head of department: prof. Ing. Miroslav Fikar, DrSc.
Workplace: Department of Information Engineering and Process Control

Topic: **Process Variable Monitoring Using Camera Data**

Language of thesis: English

Specification of Assignment:

The project focuses on the design and implementation of a method for monitoring process variables using image data from a camera. This approach, known as vision-based sensing, serves as an alternative in situations where direct measurement of a variable (e.g., particle size distribution, foam quantity, surface quality) is difficult or costly. Image data from a laboratory or industrial process will be used as a case study. The main objective is to design a system that applies image processing and machine learning methods to extract relevant information from images and convert it into an estimate of the monitored variable.

Tasks:

- Acquisition and preprocessing of image data from the process
- Design of methods for extracting relevant features from images
- Development and training of a model for estimating the monitored variable
- Validation and evaluation of the accuracy of the proposed virtual sensor
- Proposal for closed-loop implementation for control of the monitored variable

Selected bibliography:

1. M. F. Ahmed, M. K. Hasan, M. Shahjalal, M. M. Alam and Y. M. Jang, Experimental Demonstration of Continuous Sensor Data Monitoring Using Neural Network-Based Optical Camera Communications, IEEE Photonics Journal, vol. 12, no. 5, pp. 1-11, 2020, 7905011, doi: 10.1109/JPHOT.2020.3017642.

Deadline for submission of Master thesis:	15. 05. 2026
Approval of assignment of Master thesis:	27. 02. 2026
Assignment of Master thesis approved by:	prof. Ing. Miroslav Fikar, DrSc. – Study programme supervisor

Honour Declaration

I declare that the submitted master thesis was completed on my own, in cooperation with my supervisor, with the help of professional literature and other information sources, which are cited in my thesis in the reference section.

I also declare fair usage of AI and generative AI during the completion of submitted master thesis as tools for literature research, grammar and text style correction, bibliographic records generations, code understanding, LaTeX formatting and generally as consultant.

As the author of my master thesis, I declare that I did not break any third party copyrights.

Bc. Jakub
Gaborčík

Abstract

Video cameras represent a cost-effective alternative to traditional sensors for measuring intuitively perceivable process variables, such as liquid levels and volumetric flow rates. Through computer vision mechanisms and neural networks, raw visual data can be transformed into actionable engineering information.

This thesis implements deep learning techniques, specifically the “YOLO” model, to detect liquid volumes in both a simple experimental setup and a laboratory distillation column. The proposed system estimates process quantities in real-time and addresses measurement noise using a moving average filter. Finally, we discuss the potential application of a Kalman filter, alongside integration into advanced process control strategies.

The results demonstrate high precision in volume detection, while flow rate estimation provides a sufficient baseline despite hardware-related challenges in capturing droplet dynamics. The findings highlight the significant potential of camera-based sensing for industrial monitoring through environmental and model optimization.

Keywords: camera, process variables, distillation column, computer vision, deep learning, convolutional neural networks, YOLO, estimation, filter

Abstrakt

Videokamery predstavujú nákladovo efektívnu alternatívu k tradičným senzorom na meranie intuitívne vnímateľných procesných veličín, ako sú výšky hladín kvapalín a objemové prietoky. Prostredníctvom mechanizmov počítačového videnia a neurónových sietí je možné transformovať nespracované vizuálne dáta na využiteľné inžinierske informácie.

Táto diplomová práca implementuje techniky hlbokého učenia, konkrétne model „YOLO“, na detekciu objemu kvapaliny v jednoduchom experimentálnom usporiadaní aj v laboratórnej destilačnej kolóne. Navrhovaný systém odhaduje procesné veličiny v reálnom čase a rieši merací šum pomocou filtra kľzavého priemeru. Nakoniec rozoberáme potenciálnu aplikáciu Kalmanovho filtra spolu s integráciou do pokročilých stratégií riadenia procesov.

Výsledky preukazujú vysokú presnosť pri detekcii objemu, zatiaľ čo odhad prietoku poskytuje dostatočný základ napriek hardvérovým obmedzeniam pri zachytávaní dynamiky kvapiek. Zistenia zdôrazňujú významný potenciál snímania založeného na kamerách pre priemyselné monitorovanie prostredníctvom optimalizácie podmienok prostredia a parametrov modelu.

Kľúčové slová: kamera, procesné veličiny, destilačná kolóna, počítačové videnie, hlboké učenie, konvolučné neurónové siete, YOLO, odhad, filter

Contents

Honour Declaration	iii
Abstract	v
Abstrakt	vii
1 Introduction	1
2 Theoretical Background	5
2.1 AI and General Neural Networks	5
2.2 Convolutional Neural Networks	6
2.3 Data Treatment	7
2.4 State Observers	8
2.5 Distillation Column Control Strategies	10
3 Experimental Setup and Software Tools	13
3.1 Physical Setup	13
3.2 Software Tools	14
3.2.1 You Only Look Once - YOLO	14
3.2.2 Roboflow	17
3.2.3 OpenCV	17

3.2.4	HTML Visualization	17
4	Implementation	19
4.1	Preparation	19
4.2	Simple Setup Experiment	21
4.2.1	Prototyping Dataset Creation and Preparation	21
4.2.2	Prototyping Model Training	23
4.2.3	Prototyping Computation Programming and Visualization . . .	23
4.3	Laboratory Column Experiment	28
4.3.1	Laboratory Dataset Creation, Preparation and Model Training	28
4.3.2	Laboratory Computation Programming and Visualization . . .	31
5	Results	45
5.1	Simple Setup Experiment Results	45
5.2	Laboratory Experiment Results	47
5.3	Comprehensive Analysis of Distillation Column Dynamics	49
5.4	Extended concepts	53
5.4.1	Error estimation	53
5.4.2	Kalman Filter Suggestion	59
6	Conclusion	61
7	Resumé	63
	Bibliography	67

List of Figures

1.1	Schematic illustrations: a) General idea that the usage of sight can be transformed into usage of cameras in process industry, b) Reasons for using cameras instead classic sensors	2
2.1	Comparison of classical programming and machine learning concepts [6]	6
2.2	Graphical depiction of a perceptron [33]	7
2.3	Graphical depiction of how a convolutional layer in a neural network works	8
2.4	Data augmentation - different permutations of Figure 1.1b)	9
2.5	Control scheme of the distillation column reflux drum featuring a cascade configuration (LRC-FRC) for level stabilization	11
3.1	Acer Nitro ANV15-41 Notebook	14
3.2	Genius FaceCam 2000X2 Webcam	15
4.1	Setting up home experiment	22
4.2	Selected photos from all taken during simple setup experiment	22
4.3	One case of annotating in simple setup experiment	23
4.4	Distillation column with camera in school laboratory	29
4.5	Camera view at distillation column decanter with tank	29
4.6	Selected photos from all original images taken during laboratory column experiment	30

4.7	One case of annotating in laboratory setup experiment	30
5.1	Metrics for trained simple setup model: a) box precision plot, b) box recall plot, c) box mAP(50) plot, d) box mAP(50-95) plot, e) box loss plot and f) class loss plot	46
5.2	Confusion matrices for trained simple setup model a) on validation dataset, b) on testing dataset	47
5.3	Real-time volume detection visualisation of simple setup model	48
5.4	Confusion matrices for trained laboratory column model a) on validation dataset, b) on testing dataset	48
5.5	Metrics for trained laboratory setup model: a) box and mask precision plot, b) box and mask recall plot, c) box and mask mAP(50) plot, d) box and mask mAP(50-95) plot, e) box loss plot, f) segmentation loss plot and g) class loss plot	50
5.6	Real-time volume and volumetric flow detection visualisation of laboratory column model	51
5.7	Correlation between visual segmentation snapshots and the calculated time-series data for liquid volume (V) and volumetric flow rate (Q) . .	52
5.8	Measured variables of distillation column experiment	54

List of Tables

- 3.1 Technical specifications of the Acer Nitro ANV15-41 workstation [2] . 13
- 3.2 Technical specifications of the Genius FaceCam 2000X2 Webcam [25] . 16
- 3.3 YOLO Model Main Performance Metrics [45] 18

List of Equations

2.1	Kalman filter for continuous-time systems	9
2.2	Kalman filter for discrete-time systems	9
4.1	Geometric relation for cross-sectional area as function of its circumference	31
4.2	Raw volume estimation based on detected areas	31
4.3	Dynamic correction for pipe displacement	32
4.4	Submerged height of the pipes	32
4.5	Final dynamic correction for pipe displacement volume	32
4.6	Estimation of pipe height	33
4.7	Dynamic image scale calculation	33
4.8	Flow rate estimation using active zone method	33
4.9	Moving average calculation for noise reduction	33
5.1	Estimated measurement uncertainty of physical parameters measured by tape	53
5.2	Derivative of cross-sectional area with respect to circumference	53
5.3	Absolute uncertainty of cross-sectional area	55
5.4	Relative uncertainty of cross-sectional area	55
5.5	Relative uncertainties of tank and pipe cross-sectional areas	55

5.6	Estimated height ratio of pipe to tank in image	55
5.7	Estimated pipe height	56
5.8	Estimated total volume of vertical pipes	56
5.9	Positional uncertainty due to pixel quantization	56
5.10	Absolute area uncertainty based on mask perimeter	56
5.11	Relative uncertainty of the detected area	57
5.12	Statistical areas of detected components	57
5.13	Approximation for perimeter	57
5.14	Simplified relative uncertainty model	57
5.15	Calculated relative uncertainties for experiment components	57
5.16	Global YOLO performance metrics	58
5.17	Detection error rates	58
5.18	Segmentation area uncertainty	58
5.19	Total relative uncertainty	59
5.20	State-space model for flow rate estimation	60
5.21	Kalman filter update equations for flow rate	60

List of Listings

4.1	Preparation in Terminal	20
4.2	Essential imports at the beginning of every used Python script	20
4.3	Simple setup experiment - training and checking the functionality . . .	25
4.4	Simple setup experiment - volume calculation with simple CV interface - Part 1	25
4.5	Simple setup experiment - volume calculation with simple CV interface - Part 2	26
4.6	Simple setup experiment - volume calculation with simple CV interface - Part 3	27
4.7	Laboratory setup experiment - training and checking the functionality	32
4.8	Laboratory setup experiment - Python app - Part 1 - Initialization . .	34
4.9	Laboratory setup experiment - Python app - Part 2 - Image Processing	35
4.10	Laboratory setup experiment - Python app - Part 3 - Image Processing	36
4.11	Laboratory setup experiment - Python app - Part 4 - Flow calculation	37
4.12	Laboratory setup experiment - Python app - Part 5 - Smart zoom and volume calculations	38

4.13 Laboratory setup experiment - Python app - Part 6 - Moving averages	
and estimate error statistics	39
4.14 Laboratory setup experiment - Python app - Part 7 - Flask routes and	
video feed	40
4.15 Laboratory setup experiment - HTML template - Part 1 - HTML Head	
with CSS styles	41
4.16 Laboratory setup experiment - HTML template - Part 2 - Webpage	
structure	42
4.17 Laboratory setup experiment - HTML template - Part 3 - JavaScript	
Start	43
4.18 Laboratory setup experiment - HTML template - Part 4 - JavaScript End	44

List of Symbols and Abbreviations

General Symbols

Inc.	Incorporated
©	Copyright Symbol
®	Registered Trademark Symbol
™	Trademark Symbol

Latin Abbreviations

<i>e.g.</i>	“exempli gratia” (for example)
<i>etc.</i>	“et cetera” (and so on)
<i>i.e.</i>	“id est” (that is)
<i>per se</i>	“per se” (by itself, as such)

Mathematical Symbols and Operators

0	Initial
$\dot{\square}$	First derivative with respect to time
$\mathcal{N}(\mu, \sigma^2)$	Normal (Gaussian) distribution with mean μ and variance σ^2
d	Differential operator
max	Maximum value
min	Minimum value
π	Ludolph’s number
$\square^T$	Matrix transpose

$\sum$	Summation operator
d	Discrete
$f(\cdot)$	Function of an argument
f_a	Activation function
i	Summation index
N	Upper limit of summation (Total number of elements)
avg	Average

Physical Quantities

δ	Dynamic image scale	[cm/px]
ν	Sampling frequency	[1/s]
θ	Temperature	[°C]
A	Image area calculated from the segmentation mask	[px ²]
C	Circumference (Perimeter)	[cm]
D	Circle diameter	[cm]
H	Constant height	[cm]
h	Variable height	[cm]
O	Object image perimeter	[px]
P	Heating power	[kW]
p	Pressure	[Pa]
Q	Volumetric flow rate	[ml/s]
S	Physical area	[cm ²]
t	Time	[s]
V	Volume	[ml]
w	Variable object image width	[px]
y	Variable object image height	[px]

Technical Acronyms

1D	One Dimensional
2D	Two Dimensional

3D	Three Dimensional
AI	Artificial Intelligence
box_loss	Box Loss
cls_loss	Class Loss
CMOS	Complementary Metal-Oxide-Semiconductor
convnet	Convolutional Neural Network
CPU	Central Processing Unit
CV	Computer Vision
D	Distillate
DDR	Double Data Rate
dfl_loss	Distribution Focal Loss
fps	Frames Per Second
FRC	Flow Record Control
FT	Flow Transmit
Full HD	Full High Definition
Gen	Generation
GPU	Graphics Processing Unit
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
IoU	Intersection over Union
L	Reflux
l	Large
LED	Light Emitting Diode
LRC	Level Record Control
LT	Level Transmit
m	Medium
MA	Moving Average
mAP	Mean Average Precision
MIR	Mid-Infrared
MJPEG	Motion Joint Photographic Experts Group
n	Nano
NIR	Near-Infrared

OBB	Oriented Bounding Boxes
OS	Operating System
P	Precision
p	Progressive Scan
R	Recall
RAM	Random Access Memory
s	Small
SDRAM	Synchronous Dynamic Random Access Memory
seg	Instance Segmentation
seg_loss	Segmentation Loss
USB	Universal Serial Bus
UVC	Universal Video Class
v	Version
x	Extra Large
YOLO	You Only Look Once

Units

cm	Centimeter
cm ²	Centimeter squared
g	Gram
GB	Gigabyte
GHz	Gigahertz
kW	Kilowatt
ml	Milliliter
mm	Millimeter
MP	Megapixel
Pa	Pascal
px	Pixel
s	Second
°C	Degree Celsius

Parameters, vectors and matrices

$\bar{\mathbf{x}}_0$	Mean value of the initial state
ξ	Measurement noise
ξ_0	Initial state uncertainty vector
ξ_x	Process noise (White noise)
$\mathbf{A}$	System matrix (State transition matrix)
$\mathbf{B}$	Input matrix (Control matrix)
$\mathbf{C}$	Output matrix (Observation matrix)
$\mathbf{P}_0$	Initial state covariance matrix
$\mathbf{Q}$	Process noise covariance matrix
$\mathbf{R}$	Measurement noise covariance matrix
$\mathbf{u}$	Control input vector
$\mathbf{v}$	Discrete measurement noise vector
$\mathbf{w}$	Discrete process noise vector (White noise)
$\mathbf{x}$	State vector
$\mathbf{y}$	Output vector
w_i	Synaptic weight of the i -th connection (Model parameter)

Introduction

Sight is considered the most important and dominant human sense [9] - not only by the general public [8], but also by the scientific community, since the amount of data that comes from the eyes to the brain is considerably greater compared to the amount from the other senses [23], like hearing, in the same time unit [19] [20]. Therefore, there is no doubt about how essential and natural it is for people to use visual perception to get information from the environment around them, and the question might be asked: can we replicate the way we, as humans, perceive things in the process industry?

There are many significant process quantities that could potentially be measured using the visual data gathered by a camera instead of the classic sensors used in automation, like pressure sensors, temperature sensors, and so on. The main idea behind it is that if we can see it, for example, how high the level of liquid in a glass tank is, why should it not be possible to measure it by getting this information with a digital camera? By that means, we could estimate some really important quantities for the control industry this way, *e.g.*, the mentioned liquid level in a tank, its volume, even the volumetric flow rate into or from the tank. The idea of replicating human sight to obtain information via cameras is illustrated in Figure 1.1a).

Of course, why should we consider this approach at all if we do already have the mentioned classic sensors that are able to measure these quantities directly? Well, we have at least three main reasons why [24], which are graphically shown in Figure 1.1b):

- Firstly, the simplest reason would be that many industrial devices and technologies already in operation have sensors that are not digital or cannot pass on the information they are showing (simple level sensors, pneumatic sensors...). We can use a camera and image processing to collect this information somehow displayed on these sensors instead, analogically to the process of humans obtaining these data with the use of their sight. This is illustrated by the segment with the red background in Figure 1.1b).

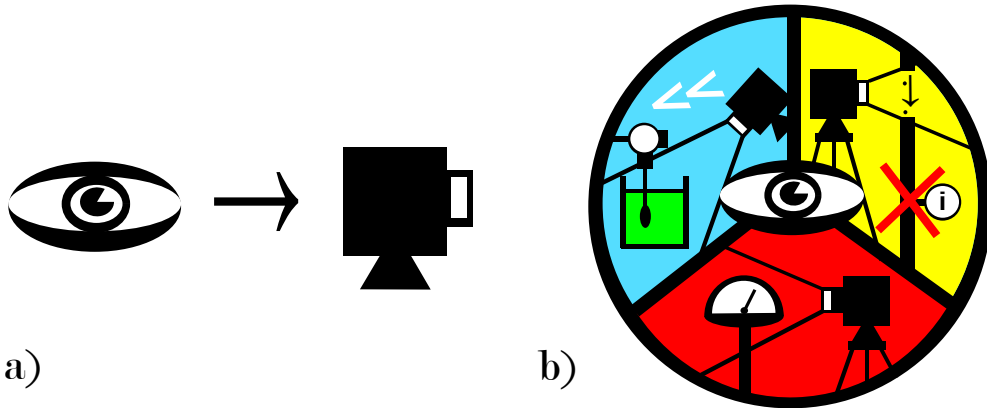


Figure 1.1: Schematic illustrations: a) General idea that the usage of sight can be transformed into usage of cameras in process industry, b) Reasons for using cameras instead classic sensors

- Secondly, we should consider attributes in which cameras might be actually better alternatives to the mentioned classic sensors, like price, durability, contamination, corrosion, *etc.* This is illustrated by the segment with the blue background in Figure 1.1b).
- And/or finally, because classic sensors simply cannot be implemented in specific systems due to difficult physical implementation, like adding a restriction into a pipe for flow measurement or because contact with a reactive chemical would destroy these classic sensors. This is illustrated by the segment with the yellow background in Figure 1.1b).

A camera by itself cannot interpret the visual data it takes. Human eyes are also only a tool for getting the information and passing it into the main processing unit - human brain. So, similarly, the next step of processing the image and extracting the wanted quantity from raw pixels is required. There are already many possible methods how to achieve this:

- Simple edge detection - algorithm finds discontinuities in intensity from one pixel to another [13].
- Projection profiles - 1D representation of a 2D image, generated by summing the pixel values along the rows (horizontal profile) or columns (vertical profile) [27].

- Simple segmentation - the process of partitioning a digital image into multiple segments to simplify its representation into a more meaningful form for analysis [48] and many others.

Yet one of the most interesting methods nowadays due to combining many of these simpler techniques and achieving great results is the use of artificial intelligence and neural networks [7].

Several existing studies have explored the applications of vision-based sensing across diverse fields. Computer vision, particularly when processed by neural networks, is a critical component of autonomous driving systems [3] and advanced security surveillance [37]. In more specialized applications, camera vision has been utilized for indirect humidity detection, where a sensor was connected to an LED that adjusted its state via pulse-width modulation [4]. In this setup, the camera's role was to monitor the operational status of the diode.

Furthermore, vision-based systems have been implemented in the wire-drawing process to detect potential operational failures in real-time [26]. In a more chemical process-related context, deep learning has been employed to identify microscopic objects and complex geometries within chemical crystal structures [1]. Other research has adapted the geometric hashing paradigm to address central problems in molecular biology, such as molecular surface recognition [31]. These techniques also find application in photometry, for instance, in developing drop-in-place computational photometers [38]. Finally, imaging technology extends beyond the visible spectrum, as demonstrated by its use in NIR/MIR dual sensors [14].

In this thesis, we specifically focus on implementing the basics of gathering process variable readings like volume and flow rate from visual data. Firstly, only by using a simple setup of a small glass cup and a syringe, and then use the tank of the distillation column located in our laboratory. Then we can suggest how to use these data in process control.

Similar goals were also pursued in other projects. "You Only Look Once", specifically "YOLOv5", which is a computer vision model based on neural networks and will also be explained later in Section 3.2.1. It was used for continuous monitoring of water levels for industrial boilers [22]. The visual data and neural networks union can be so complex that it can even reconstruct 3D scene from 2D pictures, as it was used to monitor the geometry of oil tanks [43]. There are also works about camera-based flow measurements - many of them do not do research directly on volumetric flow *per se*. But the ideas behind monitoring volumetric flow discretely as detecting falling liquid

drops are similar to traffic flow as driving cars per time unit [46], or the movement of passengers [47].

The final aim of our work is to use knowledge of camera vision and neural networks to implement a system that would be able to monitor process variables, specifically volume and volumetric flow rate, in laboratory conditions, and that can be used for distillation control, making our final master's thesis more complex than the mentioned ones.

Theoretical Background

In this chapter, we establish the theory essential for the upcoming practical part of this thesis, explaining the basic concepts of computer vision, AI, and neural networks.

2.1 AI and General Neural Networks

Artificial intelligence or AI is, by definition, the effort to automate intellectual tasks normally performed by humans, and its birth is set in the 1950s when people started to ask the question if computers could be made to “think” [6]. Then, we can define machine learning as a subset of AI with the quest to learn on its own how to perform a specific task. This concept is illustrated in Figure 2.1, showing that while in classical programming we are setting the rules and giving the system the data to get the right answers, in machine learning, it is the other way around and we let the system create its rules by giving it the wanted, right answers. This is called system training [6]. It is evident that the quality of data and answers directly influences how well the rules of the system will be set.

The fundamental concept on which the field of machine learning is built is called the perceptron [17]. It is a system shown in Figure 2.2 [33] that firstly makes a weighted summation of inputs (data) with a bias and then sends the result into a so-called activation function. An output of this system (answers) is the functional value of the activation function. The most commonly used activation functions are the binary step function, linear function, sigmoid function, and so on [17]. The role of training a machine learning model is to optimize the mentioned weights (rules) and bias to get the desired results. In machine learning terminology, we call it backpropagation [44].

We can use multiple parallel perceptrons creating a layer or even multiple layers to improve the trained results. This concept is called a neural network. Neural networks

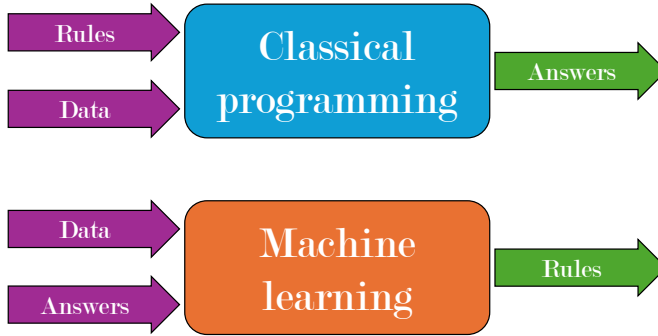


Figure 2.1: Comparison of classical programming and machine learning concepts [6]

are models with a specific structure that can learn representations from data and put an emphasis on learning successive layers of increasingly meaningful representations [6]. This is why this class of machine learning is often called deep learning. The training of neural networks behaves similarly to a simple perceptron; the main difference is the number of weights that need to be optimized and the more complex relations between each layer of perceptrons in the network [6]. The number of layers and perceptrons in them also influences the network training outcome and is the important part of the neural network architecture.

2.2 Convolutional Neural Networks

A specific type of neural network that is frequently used in computer vision applications is called a convolutional neural network or convnet. Rather than processing the raw image strictly as a high-dimensional matrix of pixel intensities, the convolutional neural network identifies and extracts hierarchical patterns, formally referred to as features. These features represent spatial characteristics of the input data, allowing the model to recognize complex geometries regardless of their absolute numerical representation in the input tensor [6]. The base of a convnet is one or more convolutional layers.

The main principle is that a filter, as a matrix of numbers representing a characteristic feature, is applied and slides through the original dataset (scanning), performing multiplication between the filter and a certain part of the data (convolution), and producing an output called a feature map, that is used in the next layers of the neural

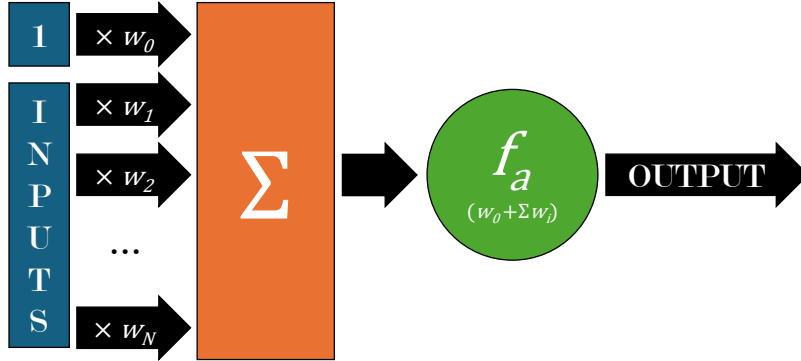


Figure 2.2: Graphical depiction of a perceptron [33]

network. The filter matrix becomes an optimized parameter. It is filled with random numbers during the first iteration of the training process (epoch), but as the iterations (epochs) go on, it forms itself into a characteristic feature. The number of filters is given by the architect of a specific network, with each filter representing a different feature, and producing multiple feature maps [6]. This principle is illustrated in Figure 2.3.

2.3 Data Treatment

As we mentioned earlier in this chapter, the quality of the trained model is directly connected to the dataset we use, meaning the higher quantity and quality of data (meaning photos or video frames in computer vision) result in better model settings and higher accuracy of the new answers we get from the model. The process of getting and preparing the dataset before the neural network training is as important as the training itself, if not more. There are many techniques to do data treatment, *e.g.*, data preprocessing and data augmentation to name a few.

Data preprocessing is a key part of the data preparation in which the raw data are converted into a format that computers and machine learning algorithms can understand and evaluate [35]. It consists of steps like resizing and picture rotation changing. For example, in the “YOLO” models that we mention in Section 3.2.1, we need to have square images as input, so steps like stretching the picture or filling blank space with black or white colour are needed to resize the original dataset [45].

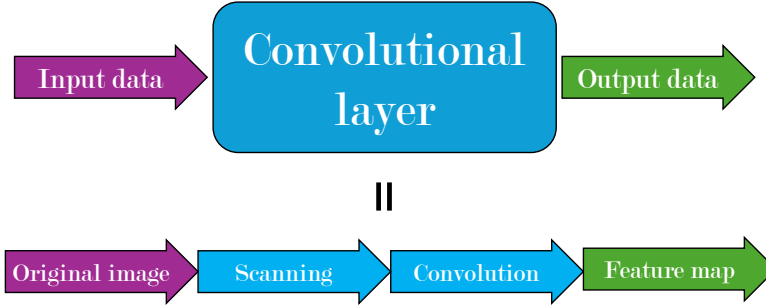


Figure 2.3: Graphical depiction of how a convolutional layer in a neural network works

Data augmentation is a method used to expand the dataset by slight changes to the original data that do not remove detected key features. For visual data, it means specifically creating permutations of existing images in a dataset to create new images [12]. The most common changes are rotation, flip, saturation, greyscale, noise addition, *etc.* Data augmentation techniques applied to the input data are illustrated in Figure 2.4, using Figure 1.1b as a reference. The decrease of picture quality by applying blur or noise doesn't necessarily mean that the quality of the dataset for training is worse – the trained model can be more robust by making some pictures more blurry or noisy. It always depends on the conditions in which it should be used in the future.

2.4 State Observers

The practical deployment of trained convolutional neural networks enables the automated extraction of critical process information. For instance, by segmenting the liquid volume within a tank or detecting individual droplets to estimate volumetric flow rates, visual data can be mapped to physical quantities using known vessel dimensions. However, such vision-based measurements are inherently susceptible to noise and detection inaccuracies, as no model achieves perfect precision under varying industrial conditions.

To mitigate these stochastic errors and enhance estimation accuracy, the system can be augmented with a simple moving average filter; alternatively, if the dynamics between the system variables are known, a state observer can be utilized. An observer is defined as a dynamic system designed such that its state converges to the state of the observed



Figure 2.4: Data augmentation - different permutations of Figure 1.1b)

deterministic system, effectively filtering out noise and measurement errors [29]. By incorporating auxiliary process variables—such as system pressure, which maintains a direct physical correlation with the volumetric flow rate—it is possible to construct a linear estimator that converges toward the true, noiseless value of the variable. One of the most established concepts for such an observer is the Kalman filter, which is defined in Equation 2.1 for continuous-time systems and Equation 2.2 for discrete-time systems. In these state-space representations, $\mathbf{x}$ denotes the state vector (e.g., flow rate), $\mathbf{A}$ represents the system matrix, and $\mathbf{C}$ is the output matrix mapping the internal state to the observed visual data $\mathbf{y}$. The terms ξ_x , $\mathbf{w}$ and ξ , $\mathbf{v}$ account for process and measurement noise, respectively. This mathematical framework allows for the recursive correction of the noisy variables obtained from the vision model, ensuring a more stable estimation of the process quantities.

$$\begin{aligned}
 \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) + \xi_x(t), & \xi_x(t) &\simeq \mathcal{N}(0, \mathbf{Q}) \\
 \mathbf{x}(0) &= \bar{\mathbf{x}}_0 + \xi_0, & \xi_0 &\simeq \mathcal{N}(0, \mathbf{P}_0) \\
 \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \xi(t), & \xi(t) &\simeq \mathcal{N}(0, \mathbf{R})
 \end{aligned} \tag{2.1}$$

$$\begin{aligned}
 \mathbf{x}_k &= \mathbf{A}_d\mathbf{x}_{k-1} + \mathbf{B}_d\mathbf{u}_{k-1} + \mathbf{w}_{k-1}, & \mathbf{w}_{k-1} &\simeq \mathcal{N}(0, \mathbf{Q}_d) \\
 \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{v}_k, & \mathbf{v}_k &\simeq \mathcal{N}(0, \mathbf{R}_d)
 \end{aligned} \tag{2.2}$$

2.5 Distillation Column Control Strategies

The filtered process variables obtained from the vision-based observer serve as primary feedback signals for higher-level automation tasks. In this context, the distillation column represents a benchmark application where precise flow and level estimations are essential for maintaining product purity through advanced control structures.

In industrial distillation processes, the reflux drum (accumulator) acts as a critical node where condensed vapor that comes from head of the column through condenser is partitioned into reflux (L) and distillate (D). To ensure operational stability and precise inventory management, a cascade control configuration is typically implemented [39]. This structure consists of two nested loops:

- Master Controller (LRC): Monitors the liquid level in the accumulator. Instead of manipulating the control valve directly, it calculates the required flow rate to maintain the setpoint level and provides this value as a reference for the inner loop.
- Slave Controller (FRC): Measures the actual flow rate and adjusts the control valve accordingly. Its primary function is to reject local disturbances, such as pressure fluctuations in the delivery line, before they impact the drum level.

The primary advantages of this multi-loop approach include superior disturbance rejection and improved overall system stability with reduced settling times compared to single-loop feedback structures [39]. The simplified control scheme, illustrating the interaction between the vision-based level measurement and the cascade control structure, is shown in Figure 2.5.

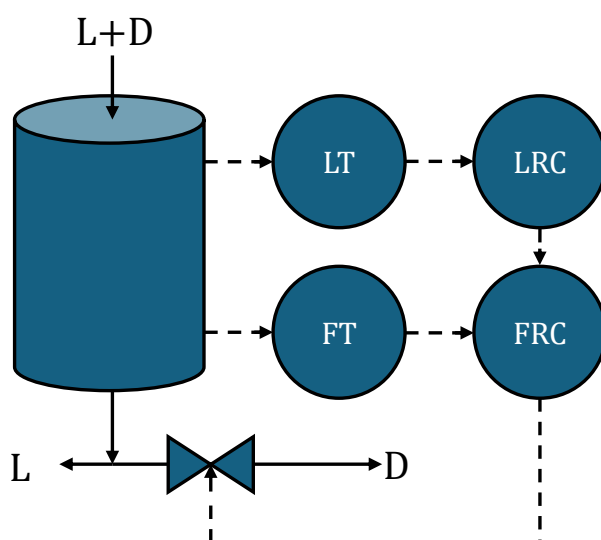


Figure 2.5: Control scheme of the distillation column reflux drum featuring a cascade configuration (LRC-FRC) for level stabilization

Experimental Setup and Software Tools

This chapter describes the physical and software setup used in this thesis, detailing the specific methods, components, and tools employed.

3.1 Physical Setup

We use a personal workstation for both the experimental and development parts of this project. The “Acer Nitro”, shown in Figure 3.1, is a great choice due to its high-performance CPU and GPU, which allow us to obtain the desired results quickly [2]. Detailed technical specifications can be seen in Table 3.1.

Table 3.1: Technical specifications of the Acer Nitro ANV15-41 workstation [2]

Parameter	Value
Manufacturer	Acer, Inc.
Model name	Nitro ANV15-41
Part number	NH.QPDEC.006
Display resolution	Full HD 1080p (up to 1920×1080 pixels)
OS	Windows 11 Home
RAM	32 GB, DDR5 SDRAM
CPU	AMD Ryzen™ 7 7735HS; 8-core & 3.20 GHz
GPU	NVIDIA® GeForce® RTX™ 4060; 8 GB
USB support	Yes (3×3.2 Gen 1 type A, 1×4)

We use the “FaceCam 2000X2 Full HD 1080P Webcam” by “Genius” for this thesis, shown in Figure 3.2. It is a simple web camera with a USB 2.0 connector that requires no complex setup; it can be plugged into a computer and used immediately [25].



Figure 3.1: Acer Nitro ANV15-41 Notebook

Official documentation mentions that the camera is compatible with “Windows” and “Mac” operating systems. However, most modern “Linux” distributions often used for microcontrollers support the USB Video Class (UVC) standard, meaning the camera can be used with them as well [41] [16]. Detailed camera specifications can be seen in Table 3.2.

3.2 Software Tools

The code for the implementation part is written in the “Python 3.12.10” programming language [34], using the popular free IDE “Visual Studio Code” [28].

3.2.1 You Only Look Once - YOLO

“You Only Look Once” (commonly referred to as “YOLO”) is a real-time object detection system based on convolutional neural networks [36]. While first introduced in 2015, several versions of the system exist today [21]. The initial versions were developed by researchers for academic purposes, but later iterations became part of the “Ultralytics” brand—these are now also known as “Ultralytics YOLO” [45].

“YOLO” offers different models with pre-made architectures and pre-trained weights



Figure 3.2: Genius FaceCam 2000X2 Webcam

for custom model training. These are categorized based on [45]:

- **Generation** - different versions based on their release date, *e.g.*, “YOLO26”, “YOLO12”, “YOLO11”, *etc.*
- **Task** - “YOLO” models generally support multiple tasks that can be performed on visual data:
 - **Classification** - labeling the entire image with one or more categories.
 - **Object detection** - identifying multiple objects in an image using **bounding boxes**.
 - **Instance segmentation** - identifying multiple objects while also determining their exact shape using **masks**.
 - Other advanced tasks include pose estimation, keypoint detection, and oriented bounding boxes (OBB).
- **Size** - based on the target device’s performance, newer “YOLO” generations usually offer several size categories: nano (n), small (s), medium (m), large (l), and extra-large (x).

We have decided to use the latest versions: “YOLO12” as the base model for the simple setup (the newest model available during the first part of the project from

Table 3.2: Technical specifications of the Genius FaceCam 2000X2 Webcam [25]

Parameter	Value
Model name	FaceCam 2000X2
Interface	USB 2.0
Sensor type	2 MP CMOS
Focus type	Fixed
Focusing range	50 mm to infinity
HD Resolution	Full HD 1080p (up to 1920×1080 pixels)
Resolution	2 MP 1920×1080 , 1280×720 , 640×480 pixels
Video frame rate	MJPEG - 1080p 25 fps; 720p 30 fps
UVC (Plug & Play)	Yes
Product dimensions	$67.8 \times 44 \times 47.8$ mm ($2.67 \times 1.73 \times 1.88$ inches)
Weight (g)	Approx. 69 g (Includes cable)

September 2025 to January 2026) and “YOLO26” as the base model for the laboratory setup (the newest model available in 2026). The object detection is sufficient for the simple setup experiment and the instance segmentation task suits our needs best since identifying the shapes of tanks, liquids, and falling drops is essential for estimating process quantities. While all model sizes could run on our workstation, the nano (n) size is key for potential implementation on microcomputers; therefore, we chose the version of the smallest size [45].

“Ultralytics YOLO” is easily integrated into a “Python” environment by installing and importing the “ultralytics” library. The specific model used is “yolo26n-seg”. Depending on the objective, three primary modes are available [45]:

- **Train** - used to train a custom model.
- **Predict** - used to test the model on data sources such as images, recorded videos, or live camera feeds.
- **Track** - similar to predict, but with the added ability to maintain object identity across frames in a video sequence.

To quantify the performance of the trained model, “YOLO” offers built-in metrics [45]. The confusion matrix provides a visual representation of the model’s classification accuracy by comparing the predicted labels against the ground truth. The diagonal

elements represent correct detections, while off-diagonal values indicate specific errors, such as misidentifying process components or confusing objects with the background. A preview of the other most common metrics (Precision, Recall, mAP, and loss functions) can be seen in Table 3.3.

mAP metrics use the Intersection over Union for their evaluation. The Intersection over Union (IoU) metric is used to quantify the overlap between the predicted segmentation mask and the ground truth annotation. It is calculated as the ratio of the area of intersection to the area of union of the two shapes. An IoU value of 1.0 indicates a perfect match, while a value of 0.5 is typically used as a threshold to distinguish between a successful detection and a false positive [45].

3.2.2 Roboflow

“Roboflow” is an online platform for visual dataset preparation, providing labeling, training, and deployment solutions for computer vision. It was developed by the company of the same name [30]. Its main purpose is to prepare “output data” before training a model by allowing users to annotate objects—marking specific items to be detected and classifying them into categories. It is compatible with “YOLO” models, supports tasks like object detection and instance segmentation, and implements data preprocessing and augmentation [12]. Additionally, if “YOLO” cannot be accessed locally, “Roboflow” offers a cloud-based solution for fast training. While “Roboflow” offers both free and paid versions, the free version was used for this thesis.

3.2.3 OpenCV

The “Open Source Computer Vision Library” (or “OpenCV”) [18] is an open-source software library for computer vision and machine learning that can be used within “Python”. It allows developers to create windows that display text, images, and video, among other functionalities. We used version 4.12.0 in this project.

3.2.4 HTML Visualization

While “OpenCV” can be used to display camera feeds with detections and process variables, web applications offer a more elegant way to visualize the output. By using libraries such as “Flask” [32] and “HTML” templates, it is possible to create a simple server for browser-based visualization.

Table 3.3: YOLO Model Main Performance Metrics [45]

1. Localization & Classification Metrics				
Interval: $\langle 0, 1 \rangle$		Dependency: their value $\rightarrow 1$ for well-performing models		
Metric	Symbol	Definition	Question	Detail
Precision	P	The proportion of true positives among all positive predictions.	How reliable is the model when it claims to see an object?	box, mask
Recall	R	The proportion of true positives among all actual positives.	How many of the actual objects did the model find?	box, mask
mAP50	mAP50	The average precision calculated across all classes at a fixed IoU threshold of 0.50.	How good is the overall performance with a loose overlap requirement?	box, mask
mAP50-95	mAP50-95	The average precision calculated across IoU thresholds from 0.50 to 0.95 (with 0.05 step).	How precise is the model in locating objects with high spatial accuracy?	box, mask
2. Loss Functions				
Interval: $\langle 0, \infty \rangle$		Dependency: their value $\rightarrow 0$ for well-performing models		
Metric	Symbol	Definition	Question	Detail
Box loss	box_loss	Coordinate error between predicted box and ground truth.	How accurately can the model locate and size the object?	box
Segmentation loss	seg_loss	Pixel-wise difference between predicted mask and actual shape.	How precise is the predicted boundary of the object?	mask
Class loss	cls_loss	Quantifies the error in predicting the correct category for a detected object.	How well can the model identify what the object is?	box, mask
Distribution focal loss	dfl_loss	Refines the boundaries of the bounding box by optimizing the probability distribution of box edges.	How well does the model handle boundary uncertainty?	box
3. Confusion matrix				

Implementation

In this chapter, we implement the theoretical knowledge in conjunction with mentioned tools to get wanted results.

4.1 Preparation

Firstly, before any experiment, there is a need to download, prepare and install all needed software, so we could start programming in “Python” using “Visual Studio Code” . We create isolated virtual environment in working folder where we install the following libraries using terminal:

- **“Ultralytics”** - to work with “YOLO”, details in Section 3.2.1
- **“Roboflow”** - to download prepared dataset, details in Section 3.2.2
- **“OpenCV”** - to process video, details in Section 3.2.3, used in Section 4.2
- **“Flask”** - as web application framework [32], details in Section 3.2.4, used in Section 4.3
- **“Matplotlib”** - as library for creating static, animated, and interactive visualizations in “Python” [15]

Then, we run test command for starting general object detection for external camera in system as source 1 and verify if “YOLO” was installed properly. Needed commands for terminal are shown in Listing 4.1.

Then, we can start each script with essential imports of these libraries, as shown in Listing 4.2.

Listing 4.1: Preparation in Terminal

```
# 1. Python version verification
> py -3.12 -m title --version

# 2. Virtual environment creation
> py -3.12 -m venv .venv

# 3. Virtual environment activation
> .venv\Scripts\activate

# 4. Essential libraries installation
(.venv) > pip install flask ultralytics roboflow matplotlib opencv-python
      ↪ -headless

# 5. YOLO Functionality test
(.venv) > yolo track model=yolo12n.pt source="1"
```

Listing 4.2: Essential imports at the beginning of every used Python script

```
1 # 1. Imports - at the beginning of every Python script
2 from flask import Flask, Response, render_template
3 # - Flask for web server and rendering
4 from flask_socketio import SocketIO
5 # - Flask-SocketIO for real-time communication
6 from ultralytics import YOLO
7 # - YOLO functionalities
8 import threading
9 # - threading for concurrent execution
10 import time
11 # - time related functions
12 import cv2
13 # - OpenCV for video processing
14 import numpy as np
15 # - scientific computing
16 import os
17 # - os for file handling
```

4.2 Simple Setup Experiment

As it was mentioned in introduction, we try to train model in simple, prototyping conditions, meaning our task is to make it detect a 200 ml glass and water and calculate the volume, all in real time.

We divided this problem into multiple tasks:

1. To create dataset we can train on
2. To train model for detecting water and glass
3. To add computation of volume and display it using “Python” and “OpenCV”

4.2.1 Prototyping Dataset Creation and Preparation

We follow these key points in order to create dataset we can train on:

1. Making photos – We want to be able to detect water and glass in our working environment, in our case on our working table, therefore we took photos of glass with different amount of water in this place, using different lighting (daylight, lamp light, dark room) and different camera angles and distances from glass. You can see how we worked in Figure 4.1 and some of the taken photos in Figure 4.2. We ended up taking 87 photos in total, starting with empty glass, adding up a little amount of water every few steps and ending up with full one.
2. Annotating object we want to detect in “Roboflow” – We use incorporated tools from hand drawing to AI object detection to annotate and classify “glasses and waters” in these photos, like you can see in example of one case in Figure 4.3.
3. Dividing data into training, validating and testing data – We set 60 images as training, 14 images as validation and 13 images as testing data randomly.
4. Data augmentation in “Roboflow” – Since we have relatively small dataset, data augmentation is needed. We check auto-orient feature and use horizontal and vertical flip, 90 degrees rotation, cropping, rotation between -15 and 15 degrees, shear, grayscale, hue, saturation, brightness, exposure, blur and noise to expand on our dataset, resulting in final dataset with 180 images in training set, 14 images in validation set and 13 images in testing set. This dataset can be accessed online (we used Instance Segmentation in “Roboflow”, but we used Object Detection task in “YOLO” as mentioned) [10].



Figure 4.1: Setting up home experiment



Figure 4.2: Selected photos from all taken during simple setup experiment

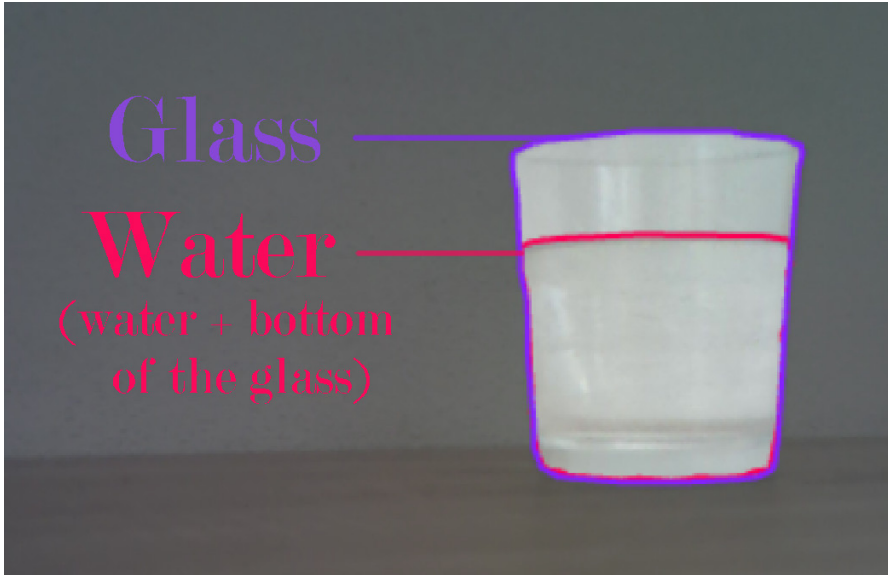


Figure 4.3: One case of annotating in simple setup experiment

4.2.2 Prototyping Model Training

We follow these key points in order to train model for detecting water and glass:

1. Downloading the dataset from “Roboflow” into computer – using generated “Python” code with unshareable api-key.
2. Training - we add function to train model with 100 epochs and with 4 cores of CPU to use. We set size to 640, since this is the size of photos automatically used in “Roboflow”. “YOLO” generates graphical results automatically.
3. Checking the functionality with tracking task with external camera in real-time.

The code can be found in Listing 4.3.

4.2.3 Prototyping Computation Programming and Visualization

We calculate ratio of glass height and water height, firstly. For full conversion into volume, we need to measure minimal ratio when glass is empty and to know the maximal

volume of glass. We assume a linear relationship between the detected liquid ratio and the actual physical volume, including a non-zero bias. This simplification is justified by the geometry of the vessel used in the prototyping phase, which approximates a cylindrical shape. The inclusion of a bias term specifically accounts for the concave bottom of the glass; this region is identified by the model as containing liquid, yet it represents a static volume that does not contribute to the incremental rise of the column. While a more complex polynomial mapping could be applied for non-standard geometries, a linear approximation was found to be sufficient for the initial validation of the measurement pipeline. We can show and update this result using “OpenCV” in real time. The implementation code is shown in Listings 4.4, 4.5 and 4.6.

Listing 4.3: Simple setup experiment - training and checking the functionality

```

18 # Simple setup experiment - training and checking the functionality
19 if __name__ == '__main__':
20     model = YOLO("yolo12n.pt") # load an official nano detection model
21     if train_var == 1:
22         #dataset download from roboflow
23         rf = Roboflow(api_key="██████████")
24         project = rf.workspace("yacomog").project("master-thesis-simple-
↪ setup-7ed3k")
25         version = project.version(2)
26         dataset = version.download("yolov12")
27         path_to_yaml = f"{dataset.location}/data.yaml"
28         #task: training
29         trained_model = model.train(data=path_to_yaml, epochs=100, imgsz
↪ =640, workers=4)
30         #task: checking the functionality with tracking task with
↪ external camera in real-time
31         trained_model_path = YOLO('runs/detect/train/weights/best.pt')
32         results = trained_model_path.track(source="1", show=True, conf
↪ =0.5) #1 is for external camera, 0 is for internal camera

```

Listing 4.4: Simple setup experiment - volume calculation with simple CV interface - Part 1

```

18 # Simple setup experiment - volume calculation with simple CV interface
19 # 1. Create a simple interface window
20 background = np.zeros((600,800,3), dtype=np.uint8)
21 cv.putText(background, 'Water_Glass_Detection_Interface', (150, 50), cv.
↪ FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
22 test_number = 5
23 # 2. Model checking with tracking task with external camera in real-time:
24 model_path = YOLO('runs/detect/train/weights/best.pt')
25 results = model_path.track(source="1", show=True, conf=0.5, stream=True)
26 dictionary = 1
27 # 3. Linear function
28 min_q = 0.28 #set from empty glass
29 a = 200/(1-min_q)
30 b = 200-a

```

Listing 4.5: Simple setup experiment - volume calculation with simple CV interface - Part 2

```

31 # 4. Real time actualization loop
32 for result in results:
33     active_frame = background.copy()
34     help_glass = help_water = used_glass = used_water = index_water =
    ↪ index_glass = 0
35     # is glass first category?
36     if dictionary == 1:
37         if result.names[0] == "Glass":
38             glass_first = 1
39         else:
40             glass_first = 0
41         dictionary = 0
42     # counting detected objects
43     objects_detected = len(result.bboxes)
44     if objects_detected == 0:
45         test_number = "error_{}_no_objects_detected"
46         volume = "error_{}_no_objects_detected"
47     if objects_detected >= 3:
48         for i in range(objects_detected):
49             if result.bboxes.cls[i] == 0:
50                 help_glass = result.bboxes.conf[i]
51                 if used_glass == 0:
52                     used_glass = help_glass
53                 if help_glass > used_glass:
54                     used_glass = help_glass
55                 index_glass = i
56             if result.bboxes.cls[i] == 1:
57                 help_water = result.bboxes.conf[i]
58                 if used_water == 0:
59                     used_water = help_water
60                 if help_water > used_water:
61                     used_water = help_water
62                 index_water = i
63             if float(result.bboxes.xywh[index_glass][3]) == 0:
64                 test_number = "error_{}_zero_division"
65                 volume = "error_{}_zero_division"
66             else:
67                 test_number = float(result.bboxes.xywh[index_water][3])/float(
    ↪ result.bboxes.xywh[index_glass][3])

```

Listing 4.6: Simple setup experiment - volume calculation with simple CV interface - Part 3

```

68 # 4. Real time actualization loop
69 if objects_detected == 2:
70     if glass_first == 0:
71         if float(result.bboxes.xywh[1][3])==0:
72             test_number = "error_{}_zero_division"
73             volume = "error_{}_zero_division"
74         else:
75             test_number = float(result.bboxes.xywh[0][3])/float(result
↪ .bboxes.xywh[1][3])
76         else:
77             if float(result.bboxes.xywh[0][3])==0:
78                 test_number = "error_{}_zero_division"
79                 volume = "error_{}_zero_division"
80             else:
81                 test_number = float(result.bboxes.xywh[1][3])/float(result
↪ .bboxes.xywh[0][3])
82 if objects_detected == 1:
83     test_number = "error_{}_only_one_object_detected"
84     volume = "error_{}_only_one_object_detected"
85 if isinstance(test_number, str):
86     volume = -1
87 elif test_number <min_q:
88     volume = 0
89 elif test_number >=1:
90     volume = 2
91 else:
92     volume = (a*test_number + b)/100
93 cv.putText(active_frame, f'Test_{}_number:{test_number}', (50, 150), cv
↪ .FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
94 cv.imshow('Water_Glass_Detection_Interface', active_frame)
95 cv.putText(active_frame, f'Volume_{}_dl:{volume:.2f}', (50, 250), cv.
↪ FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
96 cv.imshow('Water_Glass_Detection_Interface', active_frame)
97 if cv.waitKey(1) & 0xFF == ord('q'):
98     break # Ends program when 'q' key is pressed
99 # 5. End the interface when a key is pressed
100 cv.destroyAllWindows()

```

4.3 Laboratory Column Experiment

After successfully implementing the prototyping setup experiment, we can move to a more complex problem: the laboratory distillation column, which is primarily used to distill a binary mixture of water and methanol. The column itself with set camera can be seen in Figure 4.4. The camera view at column decanter with tank is shown in Figure 4.5. The documentation is referenced [5].

4.3.1 Laboratory Dataset Creation, Preparation and Model Training

Similarly to simple setup, we create a dataset with “FaceCam” by recording the decanter throughout the distillation process:

1. We start distillation column with heating power set at 0.75 kW. We record the decanter with empty tank. We are in total recycle mode, meaning all liquid is going back to the column, but using zero reflux ration using camera.
2. Upon reaching the boiling point, the distillation process initiates, resulting in the formation of droplets that descend into the decanter.
3. We close the valve under the decanter, so the tank starts to fill up with liquid and we can see the level rising.
4. After reaching maximal level because of overflow pipes, we can open the valve under the decatner to go back to normal working condition.
5. After reaching the steady state, we change the heating power to 0.9 kW and we can see how the process changes.
6. After reaching the new steady state, we end the experiment.

The experimental inputs are defined using the laboratory’s standard “MATLAB” and “Simulink” environment [42], and another output variables (excluding camera data), such as process temperatures and pressures, are recorded in this environment and used for further process control suggestions.

The raw dataset was constructed by extracting frames from the recorded video sequences at randomized intervals to ensure a diverse representation of the process states. These frames were subsequently cropped, resulting in a finalized dataset of 354 original images ready for annotation (256 in training dataset, 21 in validation dataset and 14 in testing



Figure 4.4: Distillation column with camera in school laboratory



Figure 4.5: Camera view at distillation column decanter with tank

dataset) and 803 images after augmentation (768 in training dataset, 21 in validation dataset and 14 in testing dataset), consisting of decanter pictures with empty tank, with tank filled with liquid in different levels, all with and without drops from the feed pipe, and also background-only pictures. The images can be made square in data

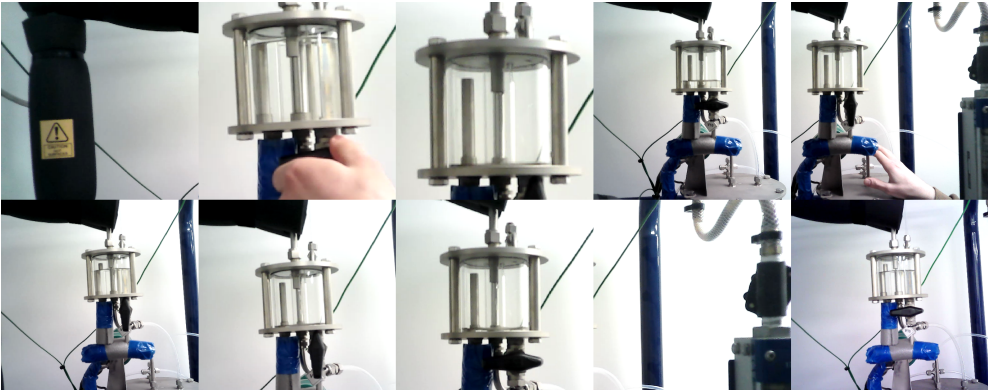


Figure 4.6: Selected photos from all original images taken during laboratory column experiment

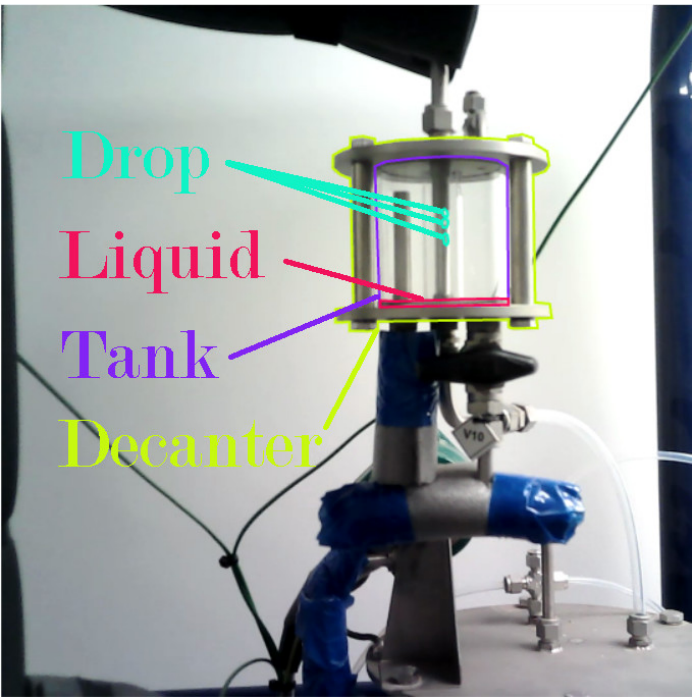


Figure 4.7: One case of annotating in laboratory setup experiment

preprocessing step, however, stretching or filling blank space would decrease the quality. Since quantity is as important, too, we tried to add and annotate as many pictures as possible. To make model robust, we work with variety of camera angles and different lightning conditions. Some of the pictures can be seen in Figure 4.6. Example of annotated picture can be seen in Figure 4.7. The dataset can be accessed online [11].

We train the model same way as in simple setup experiment. The code can be found in Listing 4.7.

4.3.2 Laboratory Computation Programming and Visualization

The software implementation for real-time monitoring of the distillation process is based on a combination of computer vision and geometric modeling. The system utilizes measured physical parameters of the experimental setup to translate pixel-based detections into meaningful physical units.

To calibrate the visual data, the following dimensions were physically measured and integrated into the configuration:

- $C_{\text{tank}} = 23.5 \text{ cm}$ – the circumference of the collection tank,
- $C_{\text{tube}} = 4.0 \text{ cm}$ – the circumference of the decanter tube,
- $H_{\text{tank}} = 7.8 \text{ cm}$ – the total measurable height of the tank.

From these measurements, the physical cross-sectional area S is derived using the geometric relation:

$$S = \frac{C^2}{4\pi} \quad (4.1)$$

The volume of the liquid in the tank is determined by the ratio of the detected liquid area to the total visible inner area of the tank (liquid, empty space, and falling drops):

$$V_{\text{raw}} = \left(\frac{A_{\text{liquid}}}{A_{\text{tank}} + A_{\text{liquid}} + A_{\text{drops}}} \cdot S_{\text{tank}} \cdot H_{\text{tank}} \right) \quad (4.2)$$

Listing 4.7: Laboratory setup experiment - training and checking the functionality

```

18 # Laboratory setup experiment - training and checking the functionality
19 if __name__ == '__main__':
20     model = YOLO("yolo26n-seg.pt") # load an official nano detection
21     ↪ model
22     # dataset download from roboflow
23     rf = Roboflow(api_key="██████████")
24     project = rf.workspace("yacomog").project("master-thesis-laboratory-
25     ↪ setup")
26     version = project.version(4)
27     dataset = version.download("yolov26")
28     path_to_yaml = f"{dataset.location}/data.yaml"
29     trained_model = model.train(data=path_to_yaml, epochs=500, imgsz
30     ↪ =1088, workers=4, batch=12, patience=50)
31     # The following path is used for tracking after training
32     trained_model_path = YOLO('runs/segment/train/weights/best.pt')
33     results = trained_model_path.track(source=1, show=True, conf=0.5,
34     ↪ imgsz=1088)

```

To achieve high precision across all filling levels, a dynamic correction for the volume displaced by the submerged portion of the pipes V_{displace} is applied. The algorithm calculates the actual volume V_{ml} as follows:

$$V_{\text{liquid}} = \max(V_{\text{raw}} - V_{\text{displace}}, 0) \quad (4.3)$$

The displacement volume V_{displace} depends on the current liquid level h_{liquid} . The submerged height of the pipes h_{displace} is constrained by their maximum visible height H_{pipe} :

$$h_{\text{displace}} = \min(h_{\text{liquid}}, H_{\text{pipe}}) \quad (4.4)$$

where h_{liquid} is the estimated height of the liquid based on the fullness percentage. Since there are 2 pipes, the final dynamic correction is then defined as:

$$V_{\text{displace}} = 2 \cdot (S_{\text{pipe}} \cdot h_{\text{sub}}) \quad (4.5)$$

H_{pipe} is estimated from ratio of the seen tank and pipe circumferences and the total tank height:

$$H_{\text{pipe}} \approx 0.75 \cdot H_{\text{tank}} \quad (4.6)$$

The flow rate Q is estimated using a dynamic “active zone” method. This method identifies the region between the highest and lowest detected drops within the tank’s inner mask. First, a dynamic scale δ is calculated to determine the physical size of a pixel based on the real tank diameter D_{tank} and its current width in the image x :

$$\delta = \frac{D_{\text{tank}}}{x_{\text{tank}}} \quad (4.7)$$

The volume of the drops is then derived by multiplying their 2D area by the depth of the stream, approximated by the decanter tube diameter D_{tube} . The resulting flow rate Q is then calculated by multiplying this volume with the segmentation sampling frequency $\nu_{\text{segmentation}}$ (or fps in technical terms):

$$Q = (A_{\text{drops inside tank}} \cdot \delta^2) \cdot D_{\text{tube}} \cdot \nu_{\text{segmentation}} \quad (4.8)$$

To ensure data stability and mitigate high-frequency noise caused by detection flickering, moving averages are applied to the raw metrics. The system maintains a sliding window of the last N samples to calculate the Moving Average (MA):

$$\text{MA} = \frac{1}{N} \sum_{i=1}^N x_i \quad (4.9)$$

The window size is heuristically set to $N = 15$ for flow rate smoothing and $N = 10$ for volume estimation. The processed data is subsequently broadcasted via “WebSockets” for real-time visualization. Implemented back-end code can be viewed in Listings 4.8 - 4.14. Implemented front-end code can be viewed in Listings 4.15 - 4.18.

Listing 4.8: Laboratory setup experiment - Python app - Part 1 - Initialization

```

68 # 2. Flask app setup
69 app = Flask(__name__)
70 socketio = SocketIO(app, cors_allowed_origins="*")
71 # 3. Config & Constants
72 model = YOLO('runs/segment/train14/weights/best.pt')
73 source = 'C:\\Users\\jakub\\Pictures\\Camera_Roll\\
    ↳ WIN_20260327_13_53_15_Pro.mp4'
74 CLASS_DECANTER, CLASS_TANK, CLASS_LIQUID, CLASS_DROP = 0, 1, 2, 3
75 class_names = {0: "Decanter", 1: "Tank", 2: "Liquid", 3: "Drop"}
76 TARGET_SIZE, SMOOTHING, ENABLE_STATS, STATS_DURATION = 1088, 0.02, True,
    ↳ 25
77 # enable stats for getting areas for error estimation
78 C_TANK, C_TUBE, H_TANK, drop_volume_factor = 23.5, 4.0, 7.8, 1
79 # 3.f Calculated Physical Constants
80 PIPE_AREA_CM2 = (C_TUBE**2) / (4 * np.pi)
81 PIPES_VOLUME_ML = 2 * (PIPE_AREA_CM2 * (H_TANK * 0.75))
82 real_tank_area = (C_TANK**2) / (4 * np.pi)
83 # 4. Global State & History
84 data_lock = threading.Lock()
85 latest_input_img = overlay_to_draw = None
86 current_metrics = {"decanter": False, "tank": False, "liquid": False, "
    ↳ drops": 0, "fps": 0, "fullness": 0, "volume": 0, "volume_mov_av":
    ↳ 0, "flow": 0, "flow_mov_av": 0}
87 flow_history, volume_history, last_mov_av_update = [], [], 0
88 stats = {"tank_areas": [], "liquid_areas": [], "drop_areas": [], "
    ↳ start_time": None, "finished": False}
89 # 5. Camera parameters
90 cam = {"x": 960, "y": 960, "s": 1920.0, "t_x": 960, "t_y": 960, "t_s":
    ↳ 1920.0, "current_t_s": 1920.0, "is_moving": False, "snapshot_x":
    ↳ 960, "snapshot_y": 960, "snapshot_s": 1920.0}
91 # 6. AI Worker function
92 def ai_worker():
93     global latest_input_img, overlay_to_draw, current_metrics, cam,
    ↳ flow_history, volume_history, last_mov_av_update
94     prev_ai_time = time.time()
95     blank_decanter = np.zeros((TARGET_SIZE, TARGET_SIZE), dtype=np.uint8)
96     blank_tank = np.zeros((TARGET_SIZE, TARGET_SIZE), dtype=np.uint8)
97     blank_drops = np.zeros((TARGET_SIZE, TARGET_SIZE), dtype=np.uint8)
98     blank_liquid = np.zeros((TARGET_SIZE, TARGET_SIZE), dtype=np.uint8)

```

Listing 4.9: Laboratory setup experiment - Python app - Part 2 - Image Processing

```

99 while True:
100     if ENABLE_STATS and stats["start_time"] is None:
101         stats["start_time"] = time.time()
102     # 6.a Get the latest image for AI processing
103     with data_lock:
104         if latest_input_img is None:
105             time.sleep(0.01)
106             continue
107         img_for_ai = latest_input_img.copy()
108         snap_x, snap_y, snap_s = cam["x"], cam["y"], cam["s"]
109         results = model.track(img_for_ai, imgsz=TARGET_SIZE, persist=True
↪ , verbose=False)[0]
110         curr_ai_time = time.time()
111         current_fps = round(1 / (curr_ai_time - prev_ai_time), 1) if (
↪ curr_ai_time - prev_ai_time) > 0 else 1
112         prev_ai_time = curr_ai_time
113         local_overlay = np.zeros((TARGET_SIZE, TARGET_SIZE, 3), dtype=np.
↪ uint8)
114         new_metrics = {
115             "decanter": False, "tank": False, "liquid": False,
116             "tank_in": False, "drops": 0, "fps": current_fps
117         }
118     # 6.b Process AI results
119     if results.masks is not None and results.bboxes is not None:
120         res_masks = results.masks.xy
121         res_bboxes = results.bboxes
122         current_scale = snap_s / TARGET_SIZE
123         best_conf, best_idx = -1, -1
124         blank_decanter.fill(0)
125         blank_tank.fill(0)
126         blank_drops.fill(0)
127         blank_liquid.fill(0)
128         area_tank, area_liquid = 0, 0
129         detections = []
130         drop_y_coords = []
131         drop_px_count = 0
132     # Start stats timer
133     if ENABLE_STATS and stats["start_time"] is None:
134         stats["start_time"] = time.time()

```

Listing 4.10: Laboratory setup experiment - Python app - Part 3 - Image Processing

```

135         for i, mask_data in enumerate(res_masks):
136             cls = int(res_boxes.cls[i].item())
137             conf = float(res_boxes.conf[i].item())
138             poly = np.array(mask_data, dtype=np.int32)
139             color = (0, 255, 0) if cls == 0 else (255, 0, 0) if cls
140             == 1 else (0, 255, 255) if cls == 2 else (255, 255, 255)
141             area_1088 = cv2.contourArea(poly)
142             original_area_px = int(area_1088 * (current_scale ** 2))
143             detections.append({'cls': cls, 'conf': conf, 'poly': poly
144             == 1, 'color': color})
145
146             if cls == CLASS_DECANTER:
147                 new_metrics["decanter"] = True
148                 cv2.fillPoly(blank_decanter, [poly], 255)
149                 if conf > best_conf: best_conf, best_idx = conf, i
150             elif cls == CLASS_TANK:
151                 cv2.fillPoly(blank_tank, [poly], 255)
152                 area_tank += area_1088
153                 if ENABLE_STATS:
154                     stats["tank_areas"].append(area_1088)
155             elif cls == CLASS_LIQUID:
156                 new_metrics["liquid"] = True
157                 cv2.fillPoly(blank_liquid, [poly], 255)
158                 area_liquid += area_1088
159                 if ENABLE_STATS:
160                     stats["liquid_areas"].append(area_1088)
161             elif cls == CLASS_DROP:
162                 new_metrics["drops"] += 1
163                 cv2.fillPoly(blank_drops, [poly], 255)
164                 # Saving Y-coordinates of drops for flow calculation
165                 drop_y_coords.extend(poly[:, 1])
166                 if ENABLE_STATS:
167                     stats["drop_areas"].append(area_1088)
168
169             # Drawing overlay
170             for det in detections:
171                 cv2.fillPoly(local_overlay, [det['poly']], det['color'])

```

Listing 4.11: Laboratory setup experiment - Python app - Part 4 - Flow calculation

```

168         # Dynamic flow calculation (active zone)
169         if new_metrics["decanter"] and area_tank > 0:
170             intersection = cv2.bitwise_and(blank_decanter, blank_tank
↪ )
171             if (np.sum(intersection > 0) / np.sum(blank_tank > 0)) >
↪ 0.70:
172                 new_metrics["tank_in"] = True
173                 if new_metrics["tank_in"] and area_tank > 0 and new_metrics["
↪ drops"] > 0:
174                     highest_drop_y = min(drop_y_coords)
175                     full_inner_mask = cv2.bitwise_or(blank_tank, blank_liquid
↪ ) if new_metrics["liquid"] else blank_tank
176                     ty, tx = np.where(full_inner_mask > 0)
177                     tank_width_px = np.max(tx) - np.min(tx) if len(tx) > 0
↪ else 1 # Dynamic pixel-to-cm scale
178                     lowest_drop_y = max(drop_y_coords)
179                     bottom_y = max(lowest_drop_y, highest_drop_y + 2)
180                     active_zone_mask = np.zeros_like(full_inner_mask)
181                     y_start, y_end = int(highest_drop_y), int(bottom_y)
182                     active_zone_mask[y_start:y_end, :] = full_inner_mask[
↪ y_start:y_end, :]
183                     drops_in_active = cv2.bitwise_and(blank_drops,
↪ active_zone_mask)
184                     drop_px_count = np.sum(drops_in_active > 0)
185                     # Physical flow calculation (cylindrical approximation)
186                     if drop_px_count > 0:
187                         D_TANK = C_TANK / np.pi
188                         D_TUBE = C_TUBE / np.pi
189                         scale_cm_per_px = D_TANK / tank_width_px
190                         area_per_px_cm2 = scale_cm_per_px ** 2
191                         drop_area_cm2 = drop_px_count * area_per_px_cm2
192                         volume_ml = drop_area_cm2 * D_TUBE
193                         flow = volume_ml * current_fps * drop_volume_factor
194                         new_metrics["flow"] = round(flow, 2)
195                     else:
196                         new_metrics["flow"] = 0
197                 else:
198                     new_metrics["flow"] = 0

```

Listing 4.12: Laboratory setup experiment - Python app - Part 5 - Smart zoom and volume calculations

```

199         # Zooming and centering logic
200         with data_lock:
201             if best_idx != -1:
202                 bx1, by1, bx2, by2 = results.bboxes.xyxy[best_idx].cpu
↪ ().numpy()
203                 bw, bh = bx2 - bx1, by2 - by1
204                 if not (0.60 < max(bw, bh) / TARGET_SIZE < 0.80):
205                     cam["t_s"] = np.clip((max(bw, bh) * current_scale
↪ ) / 0.70, 450, 1920)
206                     obj_center_x, obj_center_y = (snap_x - snap_s/2) + (
↪ bx1 + bx2)/2 * current_scale, (snap_y - snap_s/2) + (by1 + by2)/2
↪ * current_scale
207                     dist_pct = (np.sqrt((obj_center_x - cam["t_x"])**2 +
↪ (obj_center_y - cam["t_y"])**2) / cam["t_s"]) * 100
208                     if (cam["is_moving"] and dist_pct > 2.0) or (not cam[
↪ "is_moving"] and dist_pct > 5.0):
209                         cam["is_moving"], cam["t_x"], cam["t_y"] = True,
↪ obj_center_x, obj_center_y
210                         elif cam["is_moving"]: cam["is_moving"] = False
211                     else:
212                         cam["current_t_s"] = min(cam["current_t_s"] * 1.02,
↪ 1920.0)
213                         cam["t_x"], cam["t_y"], cam["t_s"] = 960, 960, cam["
↪ current_t_s"]
214         # Volume calculation
215         if area_tank > 0:
216             total_inner_area = area_liquid + area_tank +
↪ drop_px_count
217             fullness_pct = (area_liquid / total_inner_area) if
↪ total_inner_area > 0 else 0
218             new_metrics["fullness"] = min(100, int(fullness_pct *
↪ 100))
219             volume_ml_corrected = max(fullness_pct * (real_tank_area
↪ * H_TANK) - 2 * (PIPE_AREA_CM2 * min(fullness_pct * H_TANK,
↪ H_PIPE_CM)), 0)
220             new_metrics["volume"], volume_liters = round(
↪ volume_ml_corrected, 1), volume_ml_corrected / 1000
221             else: new_metrics["fullness"] = new_metrics["volume"] =
↪ volume_liters = 0

```

Listing 4.13: Laboratory setup experiment - Python app - Part 6 - Moving averages and estimate error statistics

```

222         # Moving Averages
223         flow_history.append(new_metrics["flow"])
224         if len(flow_history) > 15: flow_history.pop(0)
225         new_metrics["flow_mov_av"] = round(sum(flow_history) / len(
↪ flow_history), 2)
226         volume_history.append(volume_liters)
227         if len(volume_history) > 10: volume_history.pop(0)
228         curr_time = time.time()
229         if curr_time - last_mov_av_update >= 1:
230             new_metrics["volume_mov_av"] = round(sum(volume_history)/
↪ len(volume_history), 2) if volume_history else 0
231             last_mov_av_update = curr_time
232         else:
233             new_metrics["volume_mov_av"] = current_metrics.get("
↪ volume_mov_av", 0)
234         # Print statistics after duration
235         if ENABLE_STATS and not stats["finished"]:
236             if stats["start_time"] is not None and (time.time() - stats["
↪ start_time"]) >= STATS_DURATION:
237                 stats["finished"] = True
238                 print("\n====_STATISTICS_REPORT_====")
239                 def summarize(name, values):
240                     if len(values) == 0:
241                         print(f"{name}:_no_data")
242                         return
243                     print(f"{name}:_min={min(values)},_max={max(values)},
↪ _avg={sum(values)/len(values):.2f}")
244                 summarize("Tank_area_(px)", stats["tank_areas"])
245                 summarize("Liquid_area_(px)", stats["liquid_areas"])
246                 summarize("Drop_area_(px)", stats["drop_areas"])
247                 print("====_END_OF_REPORT_====\n")
248             with data_lock:
249                 overlay_to_draw = local_overlay
250                 cam["snapshot_x"], cam["snapshot_y"], cam["snapshot_s"] =
↪ snap_x, snap_y, snap_s
251                 for k, v in new_metrics.items():
252                     if isinstance(v, np.generic): new_metrics[k] = v.item()
253                 current_metrics.update(new_metrics)
254                 socketio.emit('update_metrics', current_metrics)

```

Listing 4.14: Laboratory setup experiment - Python app - Part 7 - Flask routes and video feed

```

255 # 7. Flask routes and video feed
256 def generate_frames():
257     global latest_input_img, overlay_to_draw, cam
258     cap = cv2.VideoCapture(source)
259     frame_dur = 1.0 / (cap.get(cv2.CAP_PROP_FPS) or 30)
260     while cap.isOpened():
261         start_time = time.time()
262         ret, frame = cap.read()
263         if not ret: cap.set(cv2.CAP_PROP_POS_FRAMES, 0); continue
264         h, w = frame.shape[:2]
265         full_sq = cv2.copyMakeBorder(frame, (w-h)//2, (w-h)-(w-h)//2, 0,
↪ 0, cv2.BORDER_CONSTANT, value=[0,0,0])
266         with data_lock:
267             for k, f in zip(["x", "y", "s"], [1, 1, 0.7]): cam[k] += (cam
↪ ["t_"+k] - cam[k]) * (SMOOTHING * f)
268             x_min, y_min = [int(np.clip(cam[k] - cam["s"]/2, 0, 1920 -
↪ cam["s"])) for k in ["x", "y"]]
269             display_img = cv2.resize(full_sq[y_min:y_min + int(cam["s"]),
↪ x_min:x_min + int(cam["s"])], (TARGET_SIZE, TARGET_SIZE))
270             latest_input_img = display_img.copy()
271             if overlay_to_draw is not None: cv2.addWeighted(
↪ overlay_to_draw, 0.4, display_img, 0.6, 0, display_img)
272             yield (b'--frame\r\nContent-Type: image/jpeg\r\n\r\n' + cv2.
↪ imencode('.jpg', display_img)[1].tobytes() + b'\r\n')
273             elapsed = time.time() - start_time
274             if elapsed < frame_dur: time.sleep(frame_dur - elapsed)
275 threading.Thread(target=ai_worker, daemon=True).start()
276 @app.route('/')
277 def index(): return render_template('index.html')
278 @app.route('/video_feed')
279 def video_feed(): return Response(generate_frames(), mimetype='multipart/
↪ x-mixed-replace;boundary=frame')
280 if __name__ == '__main__': socketio.run(app, host='0.0.0.0', port=5000)

```


Listing 4.15: Laboratory setup experiment - HTML template - Part 1 - HTML Head with CSS styles

```

1 <!DOCTYPE html>
2 <html lang="sk">
3 <head>
4   <meta charset="UTF-8"><meta name="viewport" content="width=device-
   ↪ width,initial-scale=1.0">
5   <title>Smart Camera Dashboard</title>
6   <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
7   <style>
8     body { font-family: 'Segoe UI', sans-serif; background: #1e1e2f;
   ↪ color: #fff; margin: 0; padding: 20px; display: flex; height: 100
   ↪ vh; box-sizing: border-box; }
9     .main-content { display: flex; width: 100%; gap: 20px; }
10    .panel { background: #2a2a40; border-radius: 12px; padding: 15px;
   ↪ box-shadow: 0 8px 16px rgba(0,0,0,0.3); display: flex; flex-
   ↪ direction: column; }
11    .video-panel { flex: 1.8; } .data-panel { flex: 1.2; overflow-y:
   ↪ auto; }
12    h2 { margin-top: 0; font-size: 1.2rem; border-bottom: 1px solid
   ↪ #3f3f5a; padding-bottom: 10px; }
13    .video-feed { width: 100%; aspect-ratio: 1/1; object-fit: contain
   ↪ ; background: #000; border-radius: 8px; }
14    table { width: 100%; border-collapse: collapse; margin-top: 10px;
   ↪ }
15    th, td { padding: 10px; text-align: left; border-bottom: 1px
   ↪ solid #3f3f5a; }
16    th { color: #a5a5c0; font-weight: normal; font-size: 0.9rem; }
17    td { font-size: 1.05rem; font-weight: bold; }
18    .chart-container { margin-top: 20px; background: #1e1e2f; border-
   ↪ radius: 8px; padding: 10px; flex-grow: 1; min-height: 200px; }
19    .region-bar-container { display: flex; gap: 4px; height: 30px;
   ↪ margin-top: 5px; }
20    .region-segment { flex: 1; background: #3f3f5a; border-radius: 3
   ↪ px; display: flex; align-items: center; justify-content: center;
   ↪ font-size: 0.7rem; color: #4caf50; transition: height 0.3s; }
21    .status-badge { background: #4caf50; color: white; padding: 4px 8
   ↪ px; border-radius: 4px; font-size: 0.8rem; }
22  </style>
23 </head>

```

Listing 4.16: Laboratory setup experiment - HTML template - Part 2 - Webpage structure

```

24 <body>
25   <div class="main-content">
26     <div class="panel_video-panel">
27       <h2>Smart Zoom Feed <span class="status-badge">LIVE</span></h2>
28       
29     </div>
30     <div class="panel_data-panel">
31       <h2>Current Metrics</h2>
32       <table>
33         <tr><th>Decanter / Tank</th><td id="m-status">---</td></tr>
34         <tr style="color:#4caf50;"><th>Flow (Weighted Average)</th><td><span id="m-flow">0.00</span> ml/s</td></tr>
35         <tr><th>Flow MovAv </th><td><span id="m-flow-mov-av">0.00</span> ml/s</td></tr>
36         <tr><th>Tank Level</th><td><span id="m-fullness">0</span> %</td></tr>
37         <tr><th>Volume (Real-time)</th><td><span id="m-volume">0</span> ml</td></tr>
38         <tr><th>Volume (MovAv)</th><td><span id="m-volume-mov-av">0.00</span> L</td></tr>
39         <tr><th>Processing Speed</th><td><span id="m-fps">0</span> > FPS</td></tr>
40       </table>
41       <h3 style="display:none;font-size:0.9rem;color:#a5a5c0;margin:15px 0 5px 0;">Activity in 5 Regions</h3>
42       <div class="chart-container" style="display:none;"><canvas id="flowChart"></div>
43     </div>
44   </div>

```

Listing 4.17: Laboratory setup experiment - HTML template - Part 3 - JavaScript Start

```
45 <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.0.1/  
    ↪ socket.io.js"></script>  
46 <script>  
47     var socket = io();  
48     const ctx = document.getElementById('flowChart').getContext('2d')  
    ↪ ;  
49     const flowChart = new Chart(ctx, {  
50         type: 'line',  
51         data: {  
52             labels: [],  
53             datasets: [{  
54                 label: 'Prietok (ml/s)',  
55                 borderColor: '#4caf50',  
56                 backgroundColor: 'rgba(76, 175, 80, 0.1)',  
57                 data: [],  
58                 fill: true,  
59                 tension: 0.4,  
60                 pointRadius: 0  
61             }]  
62         },  
63         options: {  
64             responsive: true,  
65             maintainAspectRatio: false,  
66             scales: {  
67                 x: { display: false },  
68                 y: {  
69                     beginAtZero: true,  
70                     grid: { color: '#3f3f5a' },  
71                     ticks: { color: '#a5a5c0' }  
72                 }  
73             },  
74             plugins: {  
75                 legend: { display: false }  
76             }  
77         }  
78     });
```

Listing 4.18: Laboratory setup experiment - HTML template - Part 4 - JavaScript
End

```

79         socket.on('update_metrics', function(data) {
80             const statusEl = document.getElementById('m-status');
81             if (data.tank_in) {
82                 statusEl.innerText = "TANK_READY";
83                 statusEl.style.color = "#4caf50";
84             } else if (data.decanter) {
85                 statusEl.innerText = "DECANTER_ONLY";
86                 statusEl.style.color = "#ff9800";
87             } else {
88                 statusEl.innerText = "EMPTY";
89                 statusEl.style.color = "#f44336";
90             }
91             document.getElementById('m-flow').innerText = parseFloat(data
↪ .flow).toFixed(2);
92             document.getElementById('m-flow-mov-av').innerText =
↪ parseFloat(data.flow_mov_av).toFixed(2);
93             document.getElementById('m-fullness').innerText = data.
↪ fullness;
94             document.getElementById('m-volume').innerText = data.volume;
95             document.getElementById('m-volume-mov-av').innerText = data.
↪ volume_mov_av;
96             document.getElementById('m-fps').innerText = data.fps;
97             const bars = document.getElementById('region-bars').children;
98             if (data.region_flows) {
99                 data.region_flows.forEach((val, i) => {
100                     bars[i].innerText = val;
101                     bars[i].style.backgroundColor = val > 0 ? 'rgba(76,
↪ 175, 80, ${Math.min(0.2 + val, 0.8)})' : "#3f3f5a";
102                 });
103             }
104             const now = new Date().toLocaleTimeString();
105             flowChart.data.labels.push(now);
106             flowChart.data.datasets[0].data.push(data.flow_mov_av);
107             if (flowChart.data.labels.length > 40) {
108                 flowChart.data.labels.shift();
109                 flowChart.data.datasets[0].data.shift();
110             }
111             flowChart.update('none');
112         });
113     </script> </body> </html>

```

Results

In this chapter, we evaluate the performance of the trained models and the functionality of the developed monitoring systems. The evaluation includes a detailed analysis of neural network metrics, the implementation of different visualization interfaces, and the processing of additional experimental data from the distillation column.

5.1 Simple Setup Experiment Results

The training progress, illustrated in Figure 5.1, shows the evolution of key metrics over 100 epochs:

- The Box Precision (a)) and Box Recall (b)) both converge toward 1, meaning the model is highly accurate in its detections and rarely misses an object. The mAP50 (c)) and mAP50-95 (d)) plots show a rapid initial increase followed by stabilization, indicating that the bounding box localization is robust.
- Furthermore, the Box Loss (e)) and Class Loss (f)) consistently decline, suggesting that the model learned the features of the simple setup without significant overtraining.

The performance of the simple setup model was also assessed using confusion matrices. As shown in Figure 5.2, the high values on the diagonal for the “Glass” and “Water” classes indicate that the model reliably distinguishes between the container and the liquid. The low values in the background column of the testing dataset (Figure 5.2b) confirm that the model produces very few false positives.

For this initial experiment, the real-time visualization was implemented using the “OpenCV” library. As seen in Figure 5.3, the script overlays the detected bounding

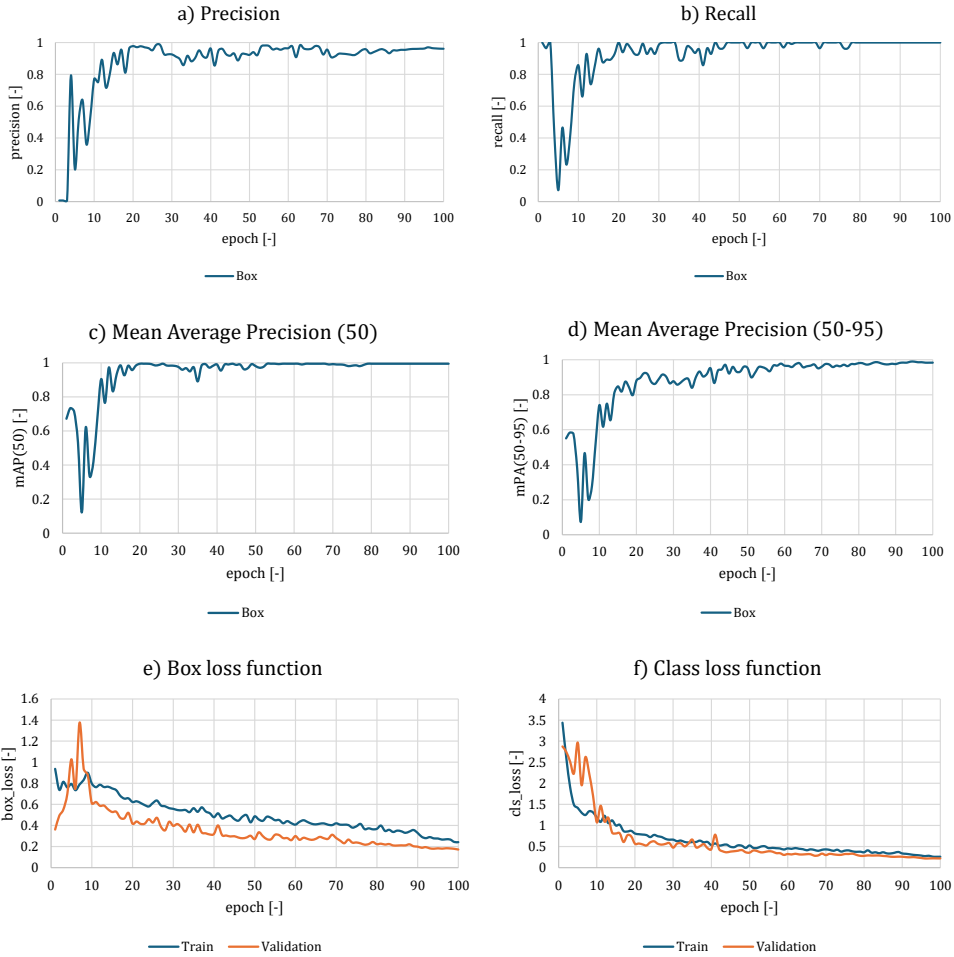


Figure 5.1: Metrics for trained simple setup model: a) box precision plot, b) box recall plot, c) box mAP(50) plot, d) box mAP(50-95) plot, e) box loss plot and f) class loss plot

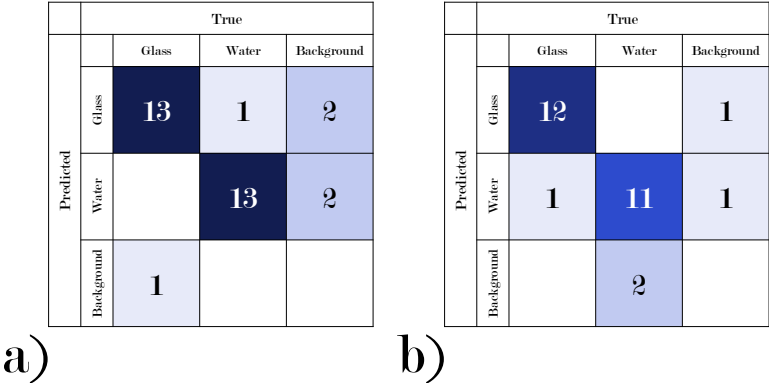


Figure 5.2: Confusion matrices for trained simple setup model a) on validation dataset, b) on testing dataset

boxes and calculated volume directly onto the video feed in an external window, providing immediate feedback on the liquid level.

5.2 Laboratory Experiment Results

The laboratory setup required a more sophisticated model capable of instance segmentation. The confusion matrices in Figure 5.4 demonstrate that even with more complex classes like “Decanter” and “Drop”, the model maintains high classification accuracy.

The training metrics for the laboratory model (Figure 5.5) provide a deeper insight into the segmentation quality:

- Box and Mask Precision (a)) and recall (b)) reflect the accuracy of both the detection boxes and the pixel-wise segmentation masks. Their convergence near 1.0 is essential for precise volume estimation.
- mAP scores (c, d)): The high mean Average Precision across different IoU thresholds confirms that the masks closely follow the actual contours of the tank and the falling drops.
- All Loss Functions (e, f, g)) decline over epochs and never begin to grow, meaning the model is not overtrained. The steady decline in Segmentation Loss

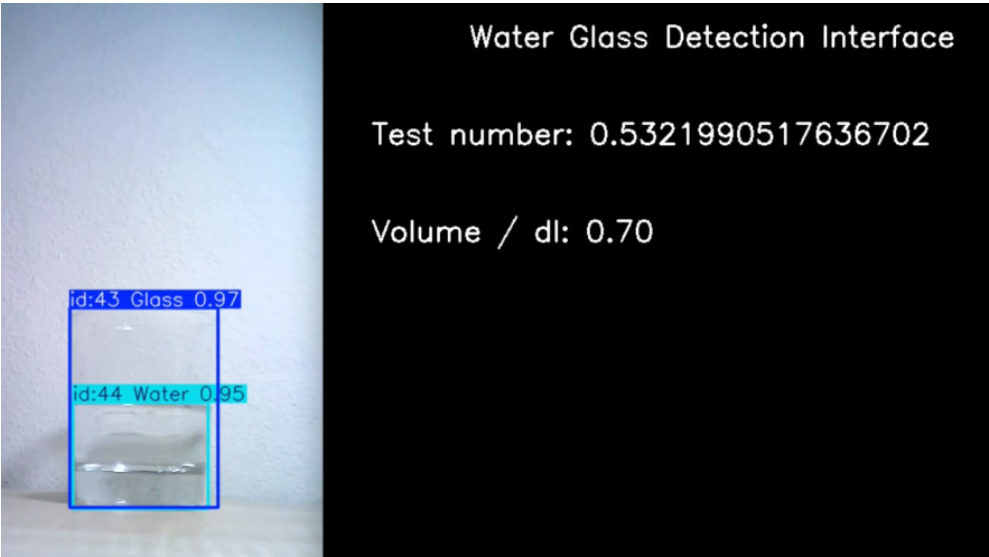


Figure 5.3: Real-time volume detection visualisation of simple setup model

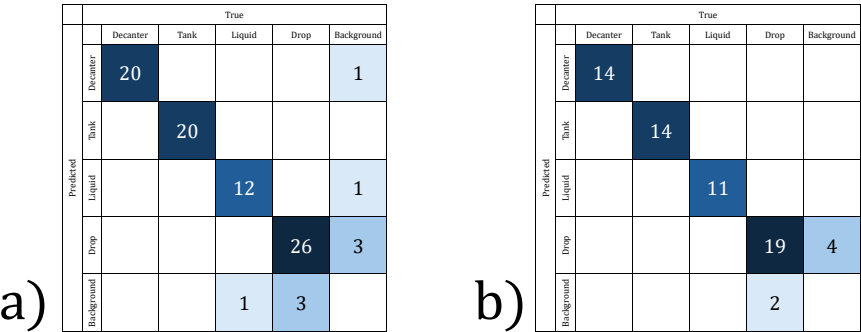


Figure 5.4: Confusion matrices for trained laboratory column model a) on validation dataset, b) on testing dataset

(f)) is particularly important, as it minimizes the error in calculating the area of the detected liquid, which directly impacts the flow rate calculation.

Unlike the prototyping setup, the laboratory monitoring system was implemented as a “Flask” web application to facilitate remote process oversight. As illustrated in Figure 5.6, this approach provides an integrated dashboard interface that displays the live video stream alongside a real-time sidebar of calculated metrics.

The central viewport visualizes the model’s inference through instance segmentation masks, which highlight the vessel geometry, the current liquid phase, and the active inflow zone. These visual detections are processed by the underlying algorithm (implemented in Section 4.3.2) to generate the quantitative data displayed in the dashboard, including the instantaneous and smoothed volumetric flow rates, the fill level percentage, and the estimated total volume. This integration effectively bridges the gap between raw neural network output and actionable engineering data, operating at a varying processing speed on the dedicated workstation.

Figure 5.7 illustrates a 30-second segment of liquid detection using the proposed model. The top section of the image displays selected snapshots from the detection process, corresponding to the liquid levels in the vessel. The liquid volume graph clearly shows that the liquid volume is detected with a high degree of precision, even in its unfiltered form (blue dots). The orange line represents the Moving Average, which successfully smooths out minor detection fluctuations. A notable phenomenon occurs in the volumetric flow rate graph during the initial phase (approximately the first 5 seconds). It is evident that as long as the droplets from the pipe are not visible—due to the outlet still being submerged by the descending liquid level—the model detects no volumetric flow ($Q = 0$ ml/s). Compared to the volume measurements, the raw flow rate calculation is significantly noisier. This noise is caused by the discrete nature of detecting individual droplets and the high sensitivity of the volume derivative to minor temporal inaccuracies in detection. In this case, applying a filter (Moving Average) is essential to obtain a readable flow rate profile, which stabilizes at approximately 5 ml/s after the initial ramp-up.

5.3 Comprehensive Analysis of Distillation Column Dynamics

In addition to the computer vision data, several process variables were recorded during the distillation experiment, as presented in Figure 5.8. These graphs provide a clear

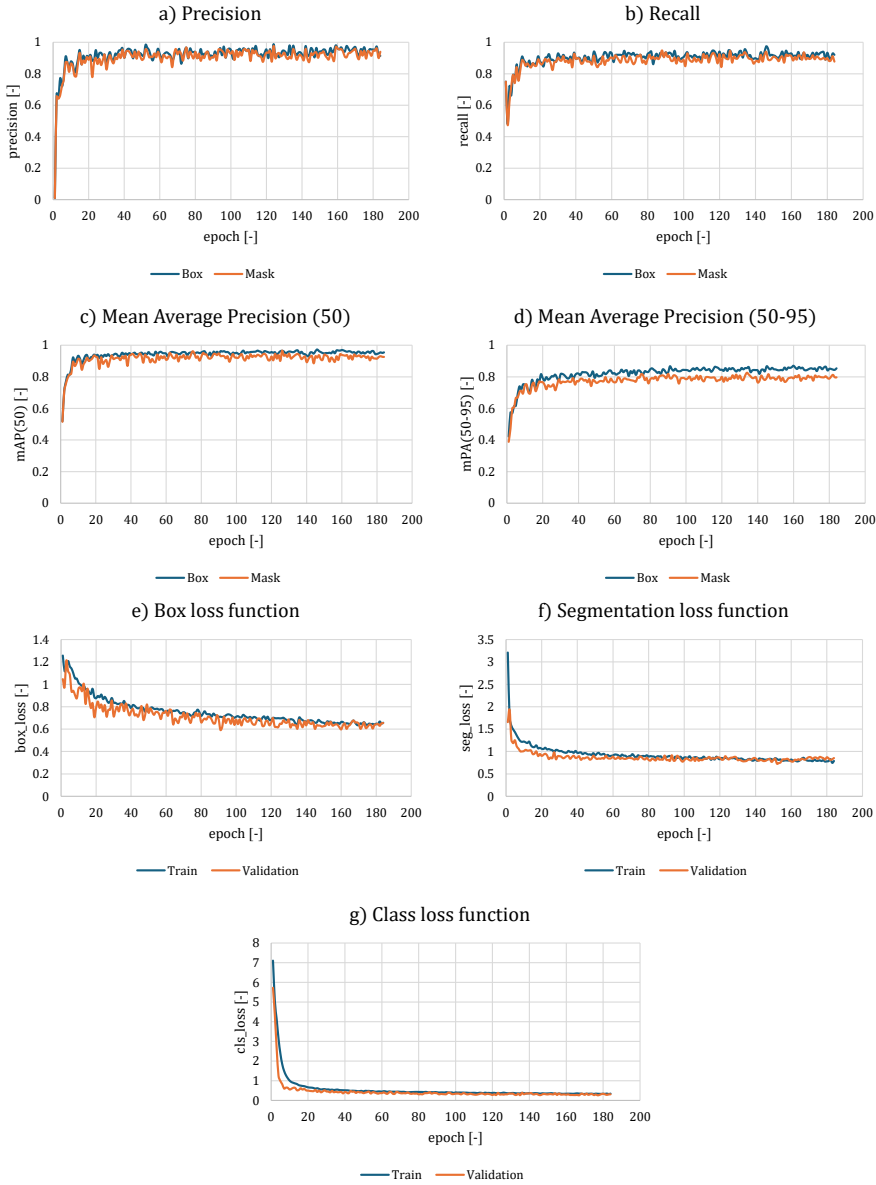


Figure 5.5: Metrics for trained laboratory setup model: a) box and mask precision plot, b) box and mask recall plot, c) box and mask mAP(50) plot, d) box and mask mAP(50-95) plot, e) box loss plot, f) segmentation loss plot and g) class loss plot

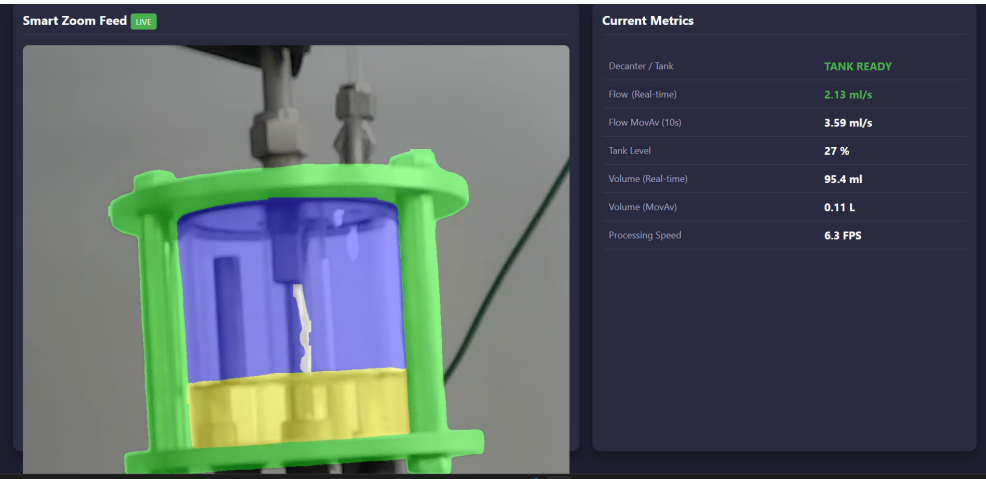


Figure 5.6: Real-time volume and volumetric flow detection visualisation of laboratory column model

timeline of the experiment:

1. Initialization phase - The early stages of the graphs (up to approximately 3000 s) show the system heating up before the actual distillation process begins.
2. Start of distillation - The sharp rise in temperatures, particularly in the reboiler and column sections (Figure 5.8b)), marks the point at which the mixture begins to vaporize and condense.
3. Power step change - In Figure 5.8a), a deliberate step change in the heating power from 0.75 kW to 0.9 kW is applied at approximately 6500 s. While this increase in energy input does not result in a visually dominant shift in reboiler pressure (Figure 5.8e)), it is physically assumed that the additional power is primarily consumed as latent heat of vaporization. This thermodynamic process leads to a higher vapor mass flow rate, which subsequently manifests as an increased distillate flow rate detected by the vision system. The relative stability of the pressure, despite the higher heating power, suggests that the system maintains a dynamic equilibrium where the condensation capacity is sufficient to process the increased vapor volume without a significant rise in internal resistance.
4. External disturbances - Notable fluctuations and “dips” in the temperature curves (Figure 5.8b)) were observed at certain intervals. These were caused by opening a window in the laboratory for air exchange to manage methanol vapors.

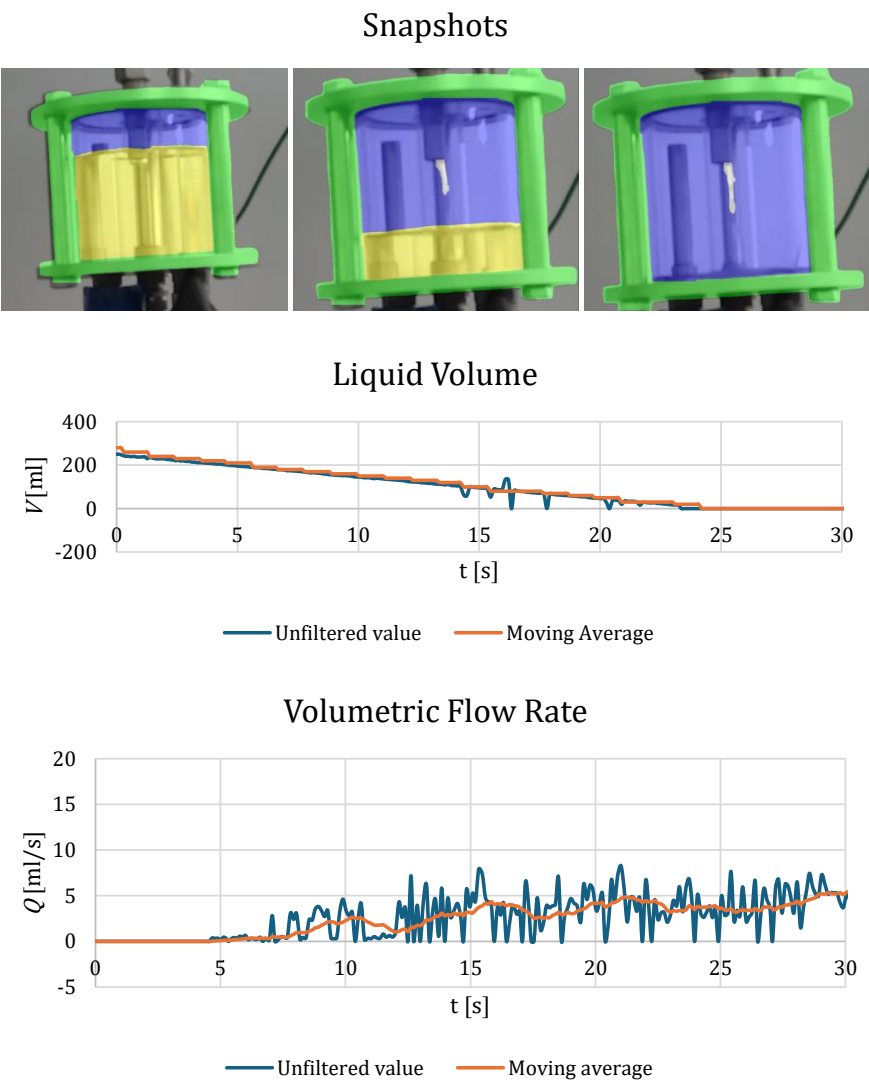


Figure 5.7: Correlation between visual segmentation snapshots and the calculated time-series data for liquid volume (V) and volumetric flow rate (Q)

These disturbances serve as an interesting test of the system's sensitivity to environmental changes.

These variables were not used directly in the neural network model, but they provide valuable context for interpreting the flow dynamics.

5.4 Extended concepts

After finishing the implementation of both experiments, we can move to more complex concepts such as error estimation, Kalman filter for smoothing the results and process control suggestions.

5.4.1 Error estimation

As mentioned in Section 4.3.2, to establish a precise mapping between the image pixels and physical units, the geometric dimensions of the tank and the decanter tube were obtained via manual measurement. The circumference of the tank C_{tank} , the circumference of the tube C_{tube} , and the measurable height of the tank H_{tank} are therefore subject to measurement uncertainty. Based on the precision of the measuring tape and repeated measurements, the uncertainty of each quantity is estimated as half of the smallest division on the tape:

$$\Delta C = \pm 0.05 \text{ cm}, \quad \Delta H = \pm 0.05 \text{ cm}. \quad (5.1)$$

The cross-sectional area of a cylindrical object is computed from its circumference using Equation 4.1.

To determine how the uncertainty in C propagates into the computed area S , we apply standard first-order uncertainty propagation [40]. The derivative of S with respect to C is:

$$\frac{dS}{dC} = \frac{C}{2\pi} \quad (5.2)$$

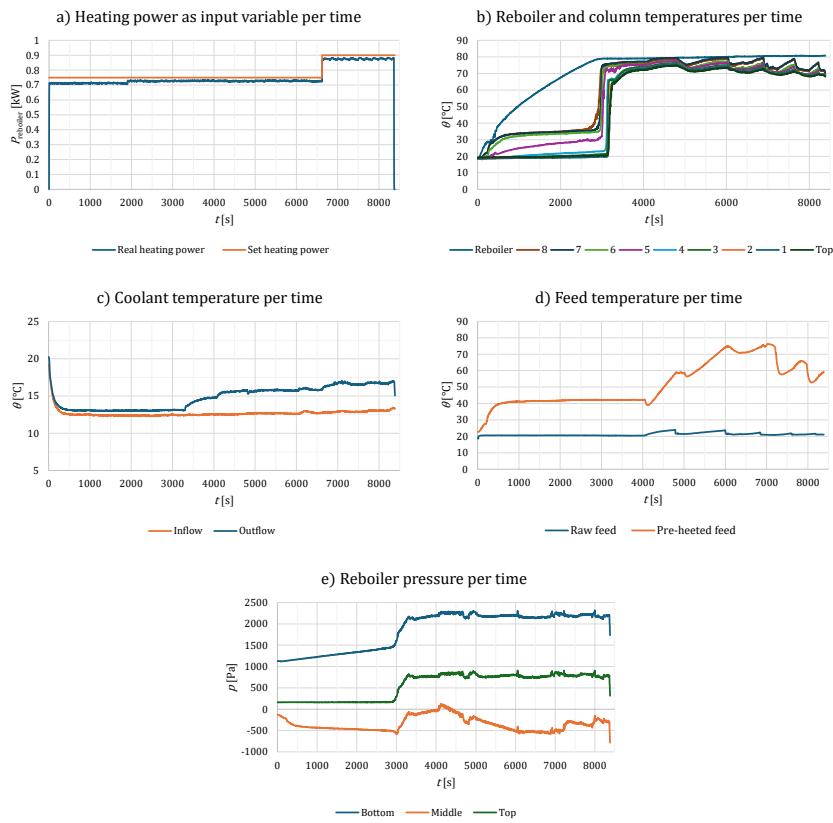


Figure 5.8: Measured variables of distillation column experiment

Thus, the absolute uncertainty of the area is:

$$\Delta S \approx \left| \frac{dS}{dC} \right| \Delta C = \frac{C}{2\pi} \Delta C \quad (5.3)$$

The corresponding relative uncertainty becomes:

$$\frac{\Delta S}{S} = \frac{2\Delta C}{C} \quad (5.4)$$

This result shows that the relative uncertainty of the computed area is twice the relative uncertainty of the measured circumference. For example, for a tank circumference of $C = 23.5$ cm and pipe circumference of $C = 4.0$ cm, the relative area uncertainty is:

$$\begin{aligned} \frac{\Delta S_{\text{tank}}}{S_{\text{tank}}} &= \frac{2 \cdot 0.05}{23.5} \approx 0.0043 \quad (0.43\%) \\ \frac{\Delta S_{\text{pipe}}}{S_{\text{pipe}}} &= \frac{2 \cdot 0.05}{4.0} \approx 0.25 \quad (2.5\%) \end{aligned} \quad (5.5)$$

This uncertainty directly propagates into the computed liquid volume and subsequently into the volumetric flow rate estimation.

Since the internal overflow pipes could not be accessed directly, their height was estimated from one image using the ratio between the visible tank height and the pipe height. From the image, the following pixel ratio was obtained:

$$H_{\text{pipe}} : H_{\text{tank}} \approx y_{\text{pipe}} : y_{\text{tank}} \approx \frac{178\text{px}}{238\text{px}} \approx 0.75 \quad (75\%) \quad (5.6)$$

Using the known physical tank height H_{tank} , the pipe diameter can be approximated as:

$$H_{\text{pipe}} \approx 0.75 H_{\text{tank}} \approx 0.75 \cdot 7.8 \text{ cm} \approx 5.85 \text{ cm} \quad (5.7)$$

Assuming there are two identical vertical pipes, their total volume is then estimated as:

$$V_{\text{pipes}} \approx 2 \cdot \frac{\pi D_{\text{pipe}}^2}{4} H_{\text{pipe}} = \frac{\pi}{2} D_{\text{pipe}}^2 H_{\text{pipe}} \quad (5.8)$$

This volume represents a constant offset that could be subtracted from the vision-based volume estimate to improve its absolute accuracy. In the current implementation, this correction is treated as an approximate systematic bias due to limited access for precise physical measurements.

Next, we move to the uncertainty caused by pixel quantization. The segmentation masks produced by the neural network are defined in a discrete pixel grid. Because the image is discretized, the boundary of each detected object cannot be localized more precisely than the size of a single pixel. The effective positional uncertainty of the object contour is approximately:

$$\Delta x = \Delta y = \pm 0.5 \text{ px} \quad (5.9)$$

The uncertainty in the area of a segmentation mask depends on its boundary length. A small uncertainty in the boundary position Δx and Δy , along the entire perimeter O_{mask} produces an absolute uncertainty in the detected area approximately equal to:

$$\Delta A \approx O_{\text{mask}} \cdot \Delta x = O_{\text{mask}} \cdot \Delta y \quad (5.10)$$

where O_{mask} is the perimeter of the mask in pixels. By substituting the quantization error $\Delta x = \Delta y = \pm 0.5 \text{ px}$, the corresponding relative uncertainty of the detected area is:

$$\frac{\Delta A}{A} \approx \frac{O_{\text{mask}} \cdot \pm 0.5 \text{ px}}{A} \quad (5.11)$$

To validate this model, a 25-second measurement run was performed on real experimental video data. During this run, the areas of all detected tank, liquid, and droplet masks were recorded (implemented in Section 4.3.2). The resulting statistics for the detected areas are:

$$\begin{aligned} A_{\text{tank}} : \min &= 1\,885.5 \text{ px}^2, \max = 153\,473.5 \text{ px}^2, \text{ avg} = 73\,099.9 \text{ px}^2 \\ A_{\text{liquid}} : \min &= 12\,624.0 \text{ px}^2, \max = 104\,896.5 \text{ px}^2, \text{ avg} = 74\,244.7 \text{ px}^2 \\ A_{\text{drops}} : \min &= 310.0 \text{ px}^2, \max = 2\,831.0 \text{ px}^2, \text{ avg} = 1\,155.1 \text{ px}^2 \end{aligned} \quad (5.12)$$

We use the approximation for the perimeter of a bounding box:

$$O_{\text{mask}} \approx 4\sqrt{A} \quad (5.13)$$

The relative uncertainty model can be simplified to:

$$\frac{\Delta A}{A} \approx \frac{4\sqrt{A} \cdot 0.5}{A} = \frac{2}{\sqrt{A}} \quad (5.14)$$

Applying this model to the measured areas, we obtain the following estimated relative uncertainties for the minimum, average, and maximum detected sizes:

$$\begin{aligned} \left(\frac{\Delta A}{A}\right)_{\text{tank}} : \min &\approx 0.5\%, \text{ avg} \approx 0.7\%, \max \approx 4.6\% \\ \left(\frac{\Delta A}{A}\right)_{\text{liquid}} : \min &\approx 0.6\%, \text{ avg} \approx 0.7\%, \max \approx 1.8\% \\ \left(\frac{\Delta A}{A}\right)_{\text{drops}} : \min &\approx 3.8\%, \text{ avg} \approx 5.9\%, \max \approx 11.4\% \end{aligned} \quad (5.15)$$

Since droplet masks are significantly smaller than tank or liquid masks, their relative uncertainty is substantially higher. This quantization noise is a major factor in the variance of the flow rate estimation and justifies the subsequent use of filtering techniques to stabilize the results.

The neural network introduces additional uncertainty due to imperfect detection and segmentation. Since the “YOLO” training logs provide overall performance metrics, the uncertainty analysis is based on the global precision, recall, and mAP scores of the model obtained from the best training epoch:

$$P \approx 0.899, \quad R \approx 0.938, \quad \text{mAP}_{50-95} \approx 0.826 \quad (5.16)$$

- Detection performance - The overall precision P represents the proportion of correct detections, while the recall R represents the proportion of correctly detected objects. These metrics quantify the probability of false positives and false negatives, respectively:

$$(1 - P) \approx 10.1\%, \quad (1 - R) \approx 6.2\% \quad (5.17)$$

Both effects contribute to uncertainty in the estimated flow rate, especially when small droplets are missed or background noise is incorrectly segmented.

- Segmentation quality - The segmentation accuracy is characterized by the mean Average Precision (mAP) for masks. The achieved mAP_{50-95} reflects the average overlap and boundary precision. This corresponds to an approximate segmentation area uncertainty of:

$$\frac{\Delta A}{A} \approx 1 - \text{mAP}_{50-95} \approx 17.4\% \quad (5.18)$$

This uncertainty primarily affects small objects such as droplets, whose area is highly sensitive to boundary errors. For large objects (tank and liquid), the relative impact is significantly smaller due to their large pixel area.

Using the global YOLO metrics from the training logs, the detection-related uncertainty

can be summarized as:

- overall false-positive rate: $\approx 10.1\%$,
- overall false-negative rate: $\approx 6.2\%$,
- segmentation uncertainty $\approx 17.4\%$,
- higher relative uncertainty for droplets due to their small size and the combination of quantization and neural network errors.

These values represent the neural-network-induced component of the total measurement uncertainty and complement the camera quantization error ($\approx 11\%$) and physical measurement uncertainty described earlier.

Assuming independence of the three uncertainty sources, the total relative uncertainty is:

$$\left(\frac{\Delta Q}{Q}\right)_{\text{total}} \approx \sqrt{\left(\frac{\Delta Q}{Q}\right)_{\text{camera}}^2 + \left(\frac{\Delta Q}{Q}\right)_{\text{physical}}^2 + \left(\frac{\Delta Q}{Q}\right)_{\text{detection}}^2}. \quad (5.19)$$

In practice, the dominant contribution comes from the neural network segmentation of droplets.

5.4.2 Kalman Filter Suggestion

The implementation of the volumetric flow rate detection showed that the raw data is significantly affected by noise, stemming from both pixel quantization and neural network uncertainties. While a moving average provided basic smoothing, a Kalman filter acting as an observer can be used to further improve the estimates and provide a more robust representation of the system state.

Since the pressures and temperatures in the distillation column are continuously monitored, the observer can be modeled with an explicit control input. The bottom reboiler pressure $p_{\text{reboiler,bottom}}$ is chosen as the input variable, as it is directly related to the generation of vapor and, consequently, serves as a driving force for the distillate flow rate. Following the discrete-time state-space representation defined in Equation 2.2, the system for flow rate estimation is modeled as:

$$\begin{aligned}\mathbf{x}_k &= \mathbf{A}_d \mathbf{x}_{k-1} + \mathbf{B}_d \mathbf{u}_{k-1} + \mathbf{w}_{k-1} \\ \mathbf{y}_k &= \mathbf{C} \mathbf{x}_k + \mathbf{v}_k\end{aligned}\tag{5.20}$$

In this formulation, the state vector $\mathbf{x}$ represents the estimated volumetric flow rate Q , the input vector $\mathbf{u}$ contains the bottom reboiler pressure p_{bot} , and $\mathbf{y}_k$ is the measurement vector obtained from the “YOLO”-based vision system. The recursive update equations for the posterior state estimate $\hat{\mathbf{x}}_{k|k}$ and the error covariance $\mathbf{P}_{k|k}$ are implemented as:

$$\begin{aligned}\mathbf{K}_k &= \mathbf{P}_{k|k-1} \mathbf{C}^T (\mathbf{C} \mathbf{P}_{k|k-1} \mathbf{C}^T + \mathbf{R}_d)^{-1} \\ \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k (\mathbf{y}_k - \mathbf{C} \hat{\mathbf{x}}_{k|k-1}) \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k \mathbf{C}) \mathbf{P}_{k|k-1}\end{aligned}\tag{5.21}$$

To fully implement this filter, it is necessary to identify the specific values of the state transition matrices $\mathbf{A}_d$ and $\mathbf{B}_d$, and to determine the noise covariance matrices $\mathbf{Q}_d$ and $\mathbf{R}_d$ through experimental calibration. The measurement noise covariance $\mathbf{R}_d$ can be estimated using the quantization and segmentation uncertainty analysis performed in previous sections. However, the exact identification of these parameters and the subsequent tuning of the filter are beyond the current scope of this work and are left for future research and development.

Conclusion

The integration of deep learning-based computer vision with classical control theory provides a comprehensive solution for distillation column monitoring and automation. The trained models achieved high precision and learning stability in segmenting and localizing critical process components, serving as a robust foundation for real-time volumetric flow rate estimation.

To further enhance the reliability of these measurements, a Kalman filter framework was proposed to fuse visual data with pressure measurements. While the mathematical structure is defined, future work will focus on the experimental determination of the noise covariance matrices $\mathbf{Q}$ and $\mathbf{R}$ to achieve optimal filter tuning.

Despite the promising results, several challenges remain for further investigation. A critical factor is the sensitivity of the vision-based system to varying lighting conditions and potential vapor condensation on glass surfaces, which can degrade segmentation accuracy. Additionally, further optimization of the inference engine is required to ensure long-term stability for high-speed process transients.

The transition from a laboratory prototyping setup to an industrial environment also raises questions regarding camera perspective and vibration compensation. Future research should address the integration of more diverse training datasets to improve the model's generalization across different vessel geometries and fluid properties, such as varying viscosity or transparency.

The real-time estimation of liquid volume and volumetric flow rate provides a foundation for implementing the cascade control strategies described in Section 2.5. Specifically, the high-frequency flow rate data obtained from the “YOLO”-based vision system could serve as the feedback signal for a Slave controller (FRC). By acting as a non-invasive soft-sensor, the system enables the rejection of supply-side disturbances in real-time.

Furthermore, the estimated volume in the decanter can be utilized by a Master controller (LRC) for inventory management. This integration demonstrates that deep learning-based measurements can effectively complement traditional industrial control strategies, bridging the gap between advanced AI perception and robust chemical process automation.

Resumé

Predkladaná práca sa podrobne zaoberá problematikou replikácie komplexného ľudského vizuálneho vnímania v špecifických a náročných podmienkach procesného priemyslu. Tento cieľ je realizovaný prostredníctvom integrácie digitálnych snímacích zariadení a pokročilých metód počítačového videnia. Hlavnou vedeckou a praktickou motiváciou pre využitie kamerových systémov ako alternatívy ku konvenčným senzorom (akými sú snímače tlaku, teploty či hladiny) je predovšetkým snaha o digitalizáciu údajov z existujúcich analógových zariadení bez nutnosti ich výmeny. Použitie optických metód prináša nesporné výhody v podobe výrazne nižších obstarávacích nákladov, vysokej odolnosti voči korozívnemu a agresívnemu chemickému prostrediu a v neposlednom rade schopnosť bezkontaktného merania v systémoch, kde je fyzická inštalácia klasických invazívnych snímačov technicky problematická. Teoretický rámec práce skúma evolúciu prístupov k spracovaniu obrazu – od elementárnych algoritmov detekcie hrán až po súčasnú špičku v podobe hlbokých konvolučných neurónových sietí, reprezentovaných najmä architektúrou „YOLO“. Autor kladie dôraz na praktickú aplikovateľnosť sledovania procesných veličín, ako sú objem a prietok, s cieľom vytvoriť robustný monitorovací systém pre laboratórnu destilačnú kolónu.

V teoretickej časti práce je umelá inteligencia definovaná ako strategický nástroj na automatizáciu kognitívnych úloh, pričom jadro výskumu spočíva v strojom učení a analýze perceptoru ako fundamentálneho stavebného prvku neurónových štruktúr. Práca podrobne dešifruje mechanizmy hlbokého učenia (deep learning) a hierarchickú architektúru konvolučných neurónových sietí (CNN). Tieto siete využívajú matematickú operáciu konvolúcie na identifikáciu vzorov v pixelových maticiach, čo je kľúčové pre extrakciu hierarchických príznakov. Osobitná pozornosť je venovaná kvalite vstupných dát; analyzujeme metódy predprípravy a augmentácie, ktoré zvyšujú robustnosť modelu voči šumu a meniacim sa priemyselným podmienkam. Keďže vizuálne merania prirodzene podliehajú stochastickým chybám, do systému je integrovaný kľavý priemer a diskutujeme o pokročilejších metódach filtrácie šumu, akými môžu byť stavové pozorovatele, napríklad Kalmanov filter. Tie zohrávajú kľúčovú úlohu pri dynamickej

filtrácii a stabilizácii odhadov procesných veličín. Teoretický prehľad končí syntézou s kaskádovým riadením, kde kamerový systém vystupuje ako súčasť spätnoväzbových regulačných slučiek, čím sa efektívne premostuje doména umelej inteligencie s klasickou procesnou automatizáciou.

Praktická realizácia riešenia sa opiera o výkonnú výpočtovú infraštruktúru vybavenú GPU „NVIDIA GeForce RTX 4060“, ktorá poskytuje potrebný paralelný výkon pre tréning a inferenciu v reálnom čase. Snímanie scény zabezpečuje Full HD kamera „Genius FaceCam 2000X2“. Celé programové prostredie je vystavané v jazyku „Python 3.12“, využívajúc synergiu knižnice „OpenCV“ pre nízkoúrovňové spracovanie obrazu a frameworku „Flask“ pre distribúciu výsledkov cez webové rozhranie. Kľúčovú rolu v detekčnom reťazci zohráva najmodernejšia iterácia systému „YOLO“ (verzie „YOLOv12“ a „YOLOv26“). Strategicky sa zameriavame na varianty „nano“ so zameraním na inštancnú segmentáciu. Tento prístup umožňuje nielen identifikáciu objektov, ale predovšetkým precízne vykreslenie pixelových masiek, čo je nevyhnutné pre presný výpočet hladín a objemov. Proces anotácie a manažmentu datasetov prebiehal v prostredí „Roboflow“, zatiaľ čo vyhodnotenie modelov sa opiera o metriky Precision, Recall, mAP a IoU.

Implementačná fáza transformuje teoretické postuláty do funkčnej programovej architektúry. Prvotná validácia prebehla formou prototypového experimentu s odmerným valcom, kde bola potvrdená schopnosť modelu detegovať rozhranie kvapaliny a lineárne aproximovať jej objem. Pri prechode na laboratórnu destilačnú kolónu vyvíjame komplexnejšie geometrické modely, ktoré konvertujú informácie z obrazových pixelov na fyzikálne jednotky pri zohľadnení optického skreslenia. Nasadená bola dynamická korekcia objemu, ktorá matematicky odčítava priestor zabieraný ponorenými vnútornými rúrkami. Výpočet prietoku je realizovaný inovatívnou metódou sledovania „aktívnej zóny“, kde algoritmus počíta segmentované kvapky dopadajúce do nádrže. Na potlačenie vysokofrekvenčných fluktuácií spôsobených pixelovou kvantizáciou boli implementované kľzavé priemery. Celý systém pracuje na princípe klient-server architektúry, kde backend vykonáva inferenciu neurónovej siete a telemetrické dáta sú distribuované cez protokol „WebSockets“.

Výsledková časť dokumentuje vysokú mieru úspešnosti navrhnutého riešenia, pričom trénované modely „YOLO“ dosiahli v oboch experimentoch stabilitu metrik a vysokú presnosť detekcie (mAP50-95 nad 0,8). V práci analyzujeme zdroje neistôt merania, ktoré rozdeľuje na fyzikálne, geometrické a algoritmické. Štatistická analýza ukázala, že najväčšiu relatívnu chybu (cca 11 – 17 %) vykazuje detekcia malých objektov, konkrétne kvapiek, čo odôvodňuje potrebu filtrácie dát. V nadväznosti na namerané dynamické trendy teploty a tlaku v kolóne autor navrhuje vylepšenie systému pomocou

„Kalmanovho“ filtra. Tento stavový pozorovateľ umožňuje fúziu obrazových dát s fyzikálnymi veličinami, čím sa eliminuje šum a zvyšuje robustnosť odhadu prietoku. V záverečnej časti prezentujeme koncept kaskádového riadenia (Level $\rightarrow$ Flow), kde výstupy z počítačového videnia slúžia ako spätná väzba pre regulátory hladiny a prietoku pre potreby stabilizácie refluxného pomeru kolóny.

Záverečné zhodnotenie práce konštatuje, že integrácia hlbokého učenia s klasickou teóriou riadenia predstavuje životaschopné riešenie pre moderný monitoring destilačných procesov. Navrhnutý systém úspešne transformuje surový video stream na presné inžinierske dáta, čím nahrádza invazívne senzory flexibilným softvérovým riešením. Hoci práca identifikuje výzvy v podobe citlivosti na svetelné podmienky, navrhnuté riešenia vytvárajú robustný rámec pre nasadenie v reálnej prevádzke. Prínos spočíva v premostení pokročilého vizuálneho vnímania umelej inteligencie s overenými metódami priemyselnej automatizácie, čo otvára dvere k ďalšej optimalizácii chemicko-technologických procesov s využitím nízkonákladových embedded platforiem.

Bibliography

- [1] Nigussie Abebe, Kebede Adane, and Berhanu Abera. Prevalence of extended-spectrum β -lactamase-producing Enterobacteriaceae in Ethiopia: a systematic review and meta-analysis. *BMC Research Notes*, 11(1):747, Oct 2018. doi: 10.1186/s13104-018-3813-8.
- [2] Acer Inc. *Acer Nitro V 15 (ANV15-41-R9XQ) Product Support and Specifications*. Acer Support, Taipei, Taiwan, 2024. Part Number: NH.QPDEC.006. [Online]. URL: <https://www.acer.com/cz-cs/support/product-support/ANV15-41/NH.QPDEC.006/details>.
- [3] Akhil Agnihotri, Prathamesh Saraf, and Kriti Rajesh Bapnad. A convolutional neural network approach towards self-driving cars. In *2019 IEEE 16th India Council International Conference (INDICON)*, pages 1–4, 2019. doi:10.1109/INDICON47234.2019.9030307.
- [4] Md. Faisal Ahmed, Moh. Khalid Hasan, Md. Shahjalal, Md. Morshed Alam, and Yeong Min Jang. Experimental demonstration of continuous sensor data monitoring using neural network-based optical camera communications. *IEEE Photonics Journal*, 12(5):1–11, 2020. doi:10.1109/JPHOT.2020.3017642.
- [5] Armfield Limited. *UOP3CC Continuous Distillation Column: Instruction Manual*. Armfield Limited, Ringwood, Hampshire, UK, issue 31 edition, October 2010. Sourced from fully sustainable forests, ISO 9001:2008 certified. URL: www.armfield.co.uk.
- [6] François Chollet. *Deep Learning with Python*. Manning Publications, Shelter Island, NY, 1st edition, 2018.
- [7] M. Egmont-Petersen, D. de Ridder, and H. Handels. Image processing with neural networks—a review. *Pattern Recognition*, 35(10):2279–2301, 2002. URL: <https://www.sciencedirect.com/science/article/pii/S0031320301001789>, doi:10.1016/S0031-3203(01)00178-9.

- [8] J. Enoch, L. Jones, and L. McDonald. Thinking about sight as a sense. *Optometry in Practice*, 21(3):2–9, 2020. © Copyright 2020. The College of Optometrists. URL: <https://www.college-optometrists.org/oip-resource/thinking-about-sight-as-a-sense.html>.
- [9] Jamie Enoch, Leanne McDonald, Lee Jones, Pete R. Jones, and David P. Crabb. Evaluating whether sight is the most valued sense. *JAMA Ophthalmology*, 137(11):1317–1320, 11 2019. arXiv:https://jamanetwork.com/journals/jamaophthalmology/articlepdf/2752217/jamaophthalmology_enoch_2019_br_190017.pdf, doi:10.1001/jamaophthalmol.2019.3537.
- [10] Jakub Gaborčík. Master thesis - simple setup - thesis version. <https://universe.roboflow.com/yacomog/master-thesis-simple-setup-7ed3k/dataset/2>, 2025. Accessed: 2026-05-09. URL: <https://universe.roboflow.com/yacomog/master-thesis-simple-setup-7ed3k/dataset/2>.
- [11] Jakub Gaborčík. Master thesis - laboratory setup - thesis version. <https://universe.roboflow.com/yacomog/master-thesis-laboratory-setup/dataset/4>, 2026. Accessed: 2026-05-09. URL: <https://universe.roboflow.com/yacomog/master-thesis-laboratory-setup/dataset/4>.
- [12] James Gallagher. What is Data Augmentation? The Ultimate Guide. <https://blog.roboflow.com/data-augmentation/>, August 2023. Accessed: 2024-05-20.
- [13] P Ganesan and G. Sajiv. A comprehensive study of edge detection for image processing applications. In *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pages 1–6, 2017. doi:10.1109/ICIIECS.2017.8275968.
- [14] A. M. Gomes, H. M. Keener, and D. L. Elwell. Development of a computer model to design and simulate windrow composting systems. *Transactions of the ASAE*, 45(3):807–819, 2002. URL: <https://elibrary.asabe.org/abstract.asp?aid=12944>, doi:10.13031/2013.8844.
- [15] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi:10.1109/MCSE.2007.55.
- [16] Ideas On Board. Linux uvc driver: Usb video class support for linux, 2020. Project homepage including the list of supported UVC devices and developer resources. Accessed: 2025-12-29. URL: <https://www.ideasonboard.org/uvc/>.
- [17] Mohammad Nazrul Islam. *Introduction to the Perceptron and Its Applications*. PhD thesis, Jahangirnagar University, 2023.

- [18] Itseez. Open source computer vision library, 2026. Accessed: 2026-01-06. URL: <https://opencv.org>.
- [19] Homer Jacobson. Information and the human ear. *The Journal of the Acoustical Society of America*, 23(4):463–471, 07 1951. arXiv:https://pubs.aip.org/asa/jasa/article-pdf/23/4/463/18729993/463_1_online.pdf, doi:10.1121/1.1906788.
- [20] Homer Jacobson. The informational capacity of the human eye. *Science*, 113(2933):292–293, 1951. URL: <https://www.science.org/doi/abs/10.1126/science.113.2933.292>, arXiv:<https://www.science.org/doi/pdf/10.1126/science.113.2933.292>, doi:10.1126/science.113.2933.292.
- [21] Peiyuan Jiang, Daji Ergu, Fangyao Liu, Ying Cai, and Bo Ma. A review of yolo algorithm developments. *Procedia Computer Science*, 199:1066–1073, 2022. The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 & 2021): Developing Global Digital Economy after COVID-19. URL: <https://www.sciencedirect.com/science/article/pii/S1877050922001363>, doi:10.1016/j.procs.2022.01.135.
- [22] Jongwon Kim, Minjun Kwon, Byeongchan So, Sewon Kim, and Hongyun So. Continuous monitoring of water levels for industrial boilers using single-stage object recognition yolov5. *International Journal of Energy Research*, 2024(1):6107765, 2024. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1155/2024/6107765>, arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1155/2024/6107765>, doi:10.1155/2024/6107765.
- [23] Kristin Koch, Judith McLean, Ronen Segev, Michael A. Freed, Michael J. Berry, Vijay Balasubramanian, and Peter Sterling. How much the eye tells the brain. *Current Biology*, 16(14):1428–1434, 2006. URL: <https://www.sciencedirect.com/science/article/pii/S0960982206016393>, doi:10.1016/j.cub.2006.05.056.
- [24] Tony R. Kuphaldt. *Lessons in Industrial Instrumentation*. Tony R. Kuphaldt / Creative Commons, version 2.14 edition, 2015. Technical means of automation textbook containing information about classic sensors. URL: https://www.ibiblio.org/kuphaldt/socratic/sinst/book/liii_2v14.pdf.
- [25] KYE SYSTEMS CORP. *FaceCam 2000X2 - Product Announcement*. Genius, No. 492, Sec. 5, Chongxin Rd., Sanchong Dist., New Taipei City 24160, Taiwan (R.O.C.), 2025. Online document with technical parameters of FaceCam 2000X2. Accessed: 2025-12-29. URL:

- https://download.geniusnet.com/WEB/Support/Product%20Announcement/Genius/06_Genius_WebCam/PA_FaceCam%202000X2_20240301.pdf.
- [26] Joakim Larsson, Anton Jansson, and Lars Pejryd. Process monitoring of the wire drawing process using a web camera based vision system. *Journal of Materials Processing Technology*, 249:512–521, 2017. URL: <https://www.sciencedirect.com/science/article/pii/S0924013617302480>, doi:10.1016/j.jmatprotec.2017.06.020.
- [27] Narges Marashi, Jamshid Shanbezadeh, and Abdolhossein Sarafzadeh. A fast and efficient skew estimation of cheque images using physical structure of image and projection profile. In *2008 15th International Conference on Mechatronics and Machine Vision in Practice*, pages 243–247, 2008. doi:10.1109/MMVIP.2008.4749541.
- [28] Microsoft. Visual studio code, 2026. URL: <https://code.visualstudio.com/>.
- [29] Ján Mikleš and Miroslav Fikar. *Process Modelling, Identification, and Control II: Identification and Optimal Control*. STU Press, Bratislava, 2004. Version: January 12, 2013. URL: <http://www.kirp.chnikf.stuba.sk/~fikar/research/other/index.htm>.
- [30] Joseph Nelson and Roboflow Team. Getting started with Roboflow, 2024. Accessed: 2026-01-06. URL: <https://blog.roboflow.com/getting-started-with-roboflow/>.
- [31] Raquel Norel, Daniel Fischer, Haim J. Wolfson, and Ruth Nussinov. Molecular surface recognition by a computer vision-based technique. *Protein Engineering*, 7(1):39–46, 01 1994. arXiv:<https://academic.oup.com/peds/article-pdf/7/1/39/4463265/7-1-39.pdf>, doi:10.1093/protein/7.1.39.
- [32] Pallets Projects. Flask documentation (stable), 2026. Version 3.0.x. URL: <https://flask.palletsprojects.com/en/stable/>.
- [33] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.
- [34] Python Software Foundation. Python programming language, 2026. 2026-01-06. URL: <https://www.python.org/>.
- [35] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, and F. Herrera. A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing*, 239:39–57, 2017. doi:10.1016/j.neucom.2017.01.078.

- [36] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016. URL: <https://arxiv.org/abs/1506.02640>, arXiv:1506.02640.
- [37] Zhenfeng Shao, Jiajun Cai, and Zhongyuan Wang. Smart monitoring cameras driven intelligent processing to big surveillance video data. *IEEE Transactions on Big Data*, 4(1):105–116, 2018. doi:10.1109/TBDATA.2017.2715815.
- [38] Sam B. Siewert, Jeries Shihadeh, Randall Myers, Jay Khandhar, and Vitaly Ivanov. Low-cost, high-performance and efficiency computational photometer design. In Jerome J. Braun, editor, *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2014*, volume 9121, page 91210G. International Society for Optics and Photonics, SPIE, May 2014. doi:10.1117/12.2049646.
- [39] Sigurd Skogestad. Dynamics and control of distillation columns: A tutorial introduction. *Chemical Engineering Research and Design*, 75(6):539–562, 1997. doi:10.1205/026387697524056.
- [40] John R. Taylor. *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements*. University Science Books, Mill Valley, California, 3rd edition, 2022.
- [41] The kernel development community. *The Linux USB Video Class (UVC) driver documentation*, 2022. Part of the official Linux Kernel documentation regarding UVC video devices. Accessed: 2025-12-29. URL: <https://www.kernel.org/doc/html/v6.1/userspace-api/media/drivers/uvccvideo.html>.
- [42] The MathWorks, Inc. Matlab and simulink, 2026. Online; accessed 04-May-2026. URL: <https://matlab.mathworks.com/>.
- [43] Ya-Lun S. Tsai, Chun-Jia Huang, Chia-Ling Chen, and Jen-Yu Han. Automatic monitoring of oil tank 3d geometry and storage changes with interferometric coherence and sar intensity information. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 17:1584–1595, 2024. doi:10.1109/JSTARS.2023.3337126.
- [44] Ultralytics. How to train a machine learning model: A comprehensive guide, 2024. Accessed: 2026-01-05. URL: <https://docs.ultralytics.com/guides/model-training-tips/>.
- [45] Ultralytics. Ultralytics YOLO documentation. <https://docs.ultralytics.com/>, 2024. Accessed: 2024-05-08.

-
- [46] Yingbin Yang, Limin Cai, Hao Wei, Tonghui Qian, and Zhengguang Gao. Research on traffic flow detection based on yolo v4. In *2021 16th International Conference on Computer Science & Education (ICCSE)*, pages 475–480, 2021. doi:10.1109/ICCSE51940.2021.9569411.
 - [47] Jinhui Zhao, Chao Li, Zhenghua Xu, et al. Detection of passenger flow on and off buses based on video images and yolo algorithm. *Multimedia Tools and Applications*, 81(4):4669–4692, 2022. doi:10.1109/10.1007/s11042-021-10747-w.
 - [48] Shiping Zhu, Xi Xia, Qingrong Zhang, and Kamel Belloulata. An image segmentation algorithm in image processing based on threshold segmentation. In *2007 Third International IEEE Conference on Signal-Image Technologies and Internet-Based System*, pages 673–678, 2007. doi:10.1109/SITIS.2007.116.