PLANET

# Facilitation of tools inspection activities

Monitor and improve the efficiency of production processes.

## Challenge

Forecast of maintenance duration for production tools: Each incoming tool is inspected before the maintenance and repair activities begin. This is done by skilled workers. The inspection results are documented according to specific levels for certain specific categories (condition of paint, dirt, missing parts, etc.). To document the condition of incoming equipment photos are taken by the worker. In order to reduce the effort for inspection and enable even non-experienced workers to conduct this task it is intended to semi-automate the inspection activity by using photos of the incoming assets and to determine their condition automatically.

## Main Requirements:

- Optimize the production planning and scheduling
- Optimize the maintenance operations based on ML and AI
- Optimize number of operators based on performance

# —— Challenge Start ——


## Student

Dimitrios Mpouziotas AM: 195

## Supervisor Professors

Marios Tyrovolas
Stylios Chrysostomos

---

**Masters Program**
Department of Computer Science and Networking
University of Ioannina

# Improving Industrial Manufacturing Production Efficiency using SOTA IM4VIS

**Facilitation of tools inspection activities**
Monitor and improve the efficiency of production processes.

# Contents

# 1 Introduction

The company faces a major challenge regarding the efficiency of tools inspections, that are causing production delays, bottlenecks in productivity, labor cost and expenses to the company. As of now, the company utilizes human labor to manually assess the quality of incoming products and resources, this significantly impacts the overall efficiency and economy of the company with the added risk of human-error. To address these limitations, the company can invest in AI-powered visual inspection systems and equipment to automate various procedures. This strategic investment will benefit the company by automating manual tasks, improving inspection quality, increasing process efficiency and improve financial impact. Consequently, this solution contributes a sustainable and easily scalable system for the company. Table 1, describes the benefits of shifting from human-labor to AI-driven manufacturing.

| Company Criteria | Human-labor | AI-driven |
|---|---|---|
| **Inspection Quality** | Low and prone to error | High and robust |
| **Process Efficiency** | Low and Inefficient | High and Automated |
| **Financial Impact** | High and expensive | Low and cheap |
| **Sustainability** | No sustainability | Energy efficient and Sustainable |
| **Scalability** | Limited by workforce | Highly scalable |

Table 1: Comparison of Human vs. AI-Driven Inspection Methods

Taking into account the description of the challenge, several issues need to be elaborated by the company to fully address the challenge and contributions of this research.

1. Should the company provide annotated data or does it require the development of a new custom dataset.
2. Does the company demand a highly accurate and real-time model to assess and monitor the efficiency of the production process? Achieving both high accuracy and performance has been a major challenge for many SOTA computer vision models.
3. Does the company demand a fully sustainable system without any human interaction for quality inspection and production efficiency monitoring?
4. What kind of equipment will be utilized to monitor the efficiency? From a wide range of cameras, thermal cameras, LIDAR, speed sensors and more?
5. Does the company demand an implementation of a Digital Twin system to monitor in real-time the production process digitally?

Additionally, this study introduces an innovative solution to address the company's challenge and demands to improve the production efficiency of products. Furthermore, the SOTA [1] system namely IM4VIS (Industrial Manufacturing 4.0 Visual-based Inspection System) facilitates computer vision methods to effectively detect defects, count objects and more.

## 1.1 Study Approach

The challenge approach begins by analyzing the problem, understanding the solution and researching various innovative technologies to implement the workflow of the company. Figure 1, illustrates this workflow design from problem analysis to Implementation and measurement in the impact of the AI-driven design.

[1] state-of-the-art: Referring to a model, system, algorithm or AI that performs best in the given market or latest technology

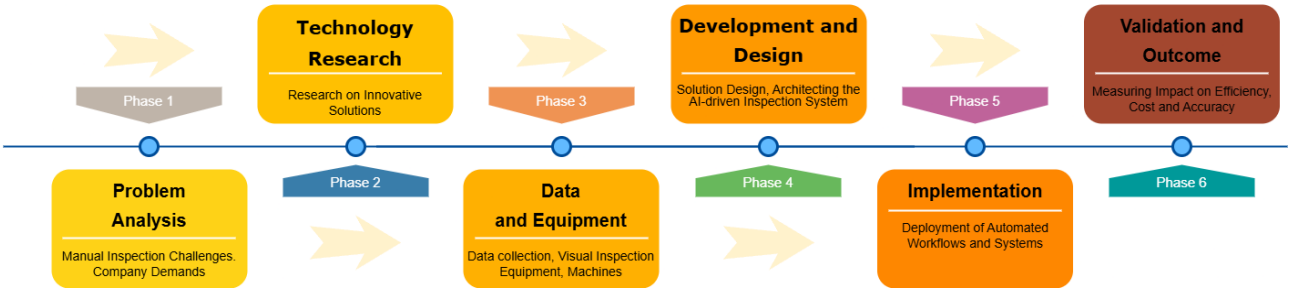



Figure 1: Workflow design describing the challenge design

The next sections describe various solutions utilized in various literature to produce an AI-driven facility for tool or resource inspection.

# 2 Technological Research

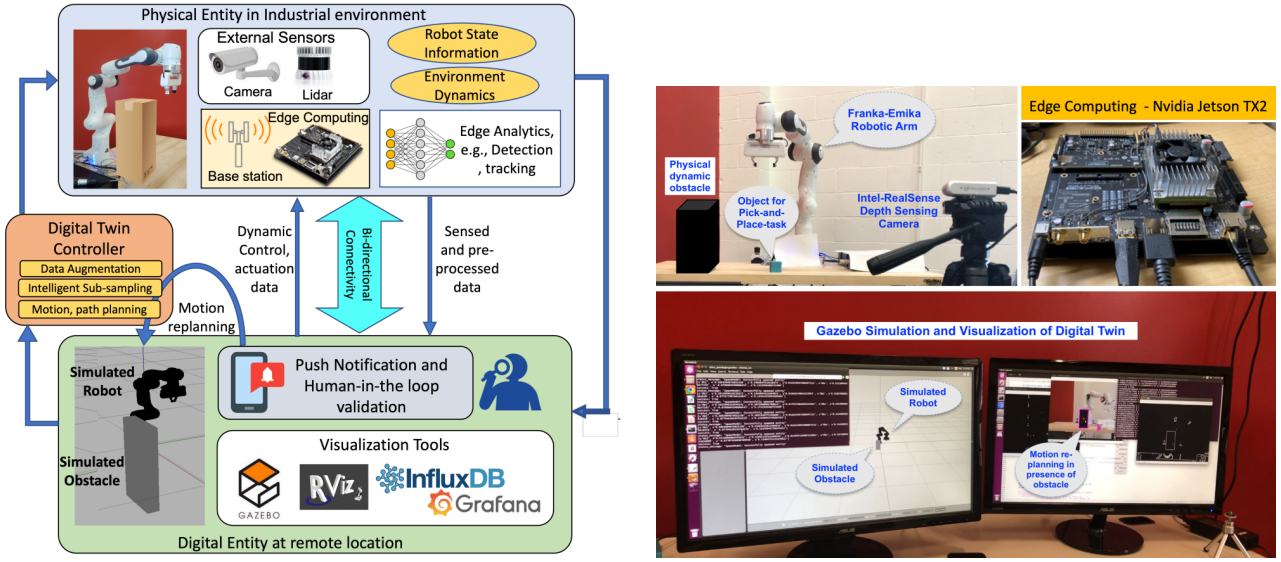
To improve the efficiency of the production process, this research expands on a collection of innovative solutions, studies and methodologies. The topics explored are as follows:

1. **Digital Twins**: By incorporating real-world data into a digital environment it enable us to remotely monitor in real-time data of the given environment.
2. **IoT Sensors**: IoT sensors are the lowest stage that is utilized to acquire valuable data in real-time. Micro-controllers capable of connecting to the wifi are able to transmit data in a cloud server such as Thinger.io [1].
3. **Computer Vision**: Using SOTA (State of the art) computer vision models can significantly improve and automate the monitoring process of incoming tools and equipment. Computer Vision allows for quality inspections, flow monitoring in conveyor belts and more.

## 2.1 Digital Twins

Digital Twins are advanced digital replicas of physical environments that aim continuously provide real-time data. Engineers are able to observe the data remotely through the cloud. At its core, Digital Twins rely on Internet of Things (IoT) devices, which enables the collection of data from sensors and embedded systems within the physical environment. Using IoT devices we can collectively transmit data using the wifi into a cloud-based system transmitting data such as temperature, gas, coordinates of objects and more.

Regarding robotics and industrialism, Sumit K. Das et al. [2] developed a specialized Digital Twin incorporating industrial IoT devices such as cameras and LIDAR (Light Detection and Ranging) [3] (Figure 2). By combining all these technologies with a robotic arm, the study developed an obstacle avoidance algorithm using the digital twin.




(a) Design of a Digital Twin for a robotic arm.

(b) Demo setup of the Digital Twin system.

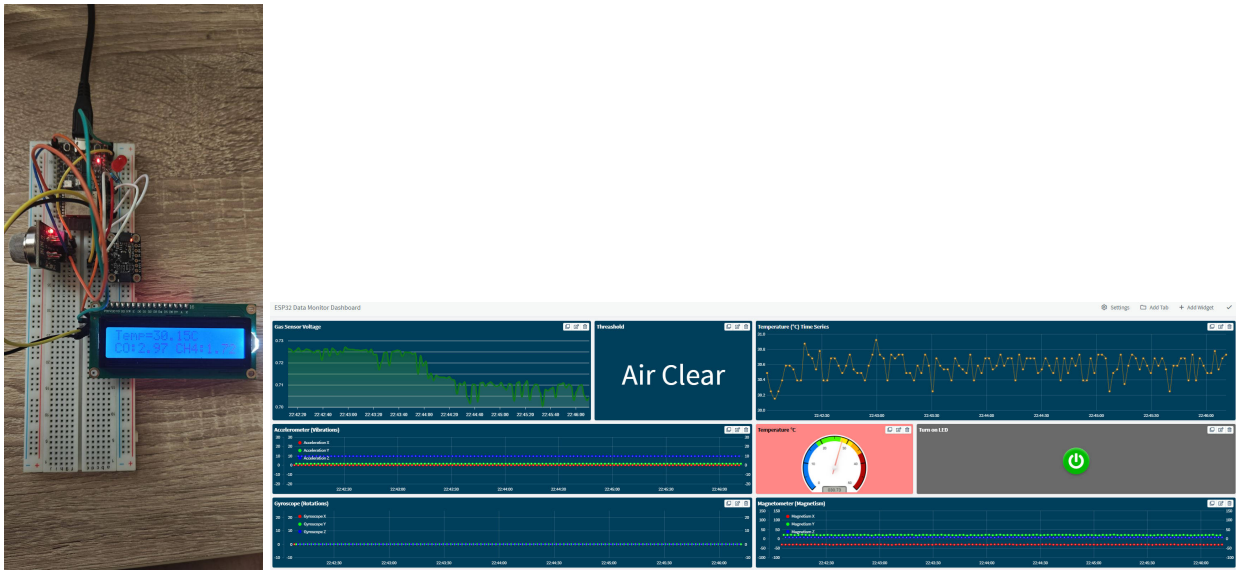
Figure 2: Architecture of a robotic Digital Twin.

## 2.2 IoT Devices

By deploying IoT Devices based on the layout and scale of the facility, manufacturers can monitor data in real-time with the use of IoT sensors. Sensors enables us to collect data from equipment such as tools, machinery or environmental parameters. Many IoT-based systems utilize micro-controller devices to process data directly on-site and transmit it in the cloud, in short this is called Edge Computing, it drastically improves the workflow of any IoT-based system. Particularly for the current challenge, these methods ensure an optimized production flow and an proactive detection of possible defects in equipment. Furthermore, manufacturers and engineers can access the cloud data retrieved by sensors in real time remotely and respond accordingly to any changes in the system. Various monitoring sensors for manufacturing include:

- **Cameras**: Cameras constantly retrieve visual data of the environment, from images to videos. There
- **Multispectral Cameras**: They enable the retrieval of various spectrums of visual information such as temperature, gas and more. Often utilized in thermal imagery mapping, satellite imagery and more.
- **LIDAR**: State of the art technology that calculates the time it took for a laser to travel from one point back to its sensor. This technology is often used for 3D mapping, measure distances or automated vehicle driving.
- **Accelerometers**: 3 dimensional (X/Y/Z) measurement of the sensor's acceleration. Often utilized to measure vibration of devices, servers and machinery in factories. These sensors are utilized for proactive defect detection in equipment.
- **Micro-controllers**: Directly communicate with surrounding sensors to process and transmit data in the cloud. In computer vision, a micro-controller equipped with a GPU (e.g. Nvidia Jetson TX2 [2]) significantly increases performance of CNN models compared to standard micro-controllers (e.g. ESP-32 [4]).

Figure 3, illustrates an IoT design that actively collects temperature, gas and motion tracking data (gyroscope, acceleration, magnetometer) and transmits it through the wifi into a cloud-based system. This design enables for real-time monitoring of data in environments.




(a) Deployed IoT sensor node with LCD readout

(b) Live monitoring on Thinger.io dashboard [1]

Figure 3: IoT environmental monitoring setup: (a) Hardware sensor prototype, (b) Cloud-based real-time visualization. Source: MSc IoT Assignment

## 2.3 Computer Vision

The integration of computer vision and image processing techniques into tool inspection tasks can significantly enhance the scope and accuracy of data collection. This challenge's goal is to utilize state-of-the-art models performing in real-time. In recent years, various advancements and innovations in technology around Artificial Intelligence have emerged. One major advancement was the introduction of Convolutional Neural Networks [5], [6] with many recent innovations including [7], Resnet [8], YOLO (You Only Look Once) [9], DeTr (Detection Transformer) [10] and ViTs (Vision Transformers) [11] and finally Co-Detr. The standard CNN architecture consists of four major parts: Input, Dimensionality Reduction, Feature Extraction and Classification, Figure 4, illustrates the standard YOLO [9] CNN based on the GoogLeNet [12] backbone.

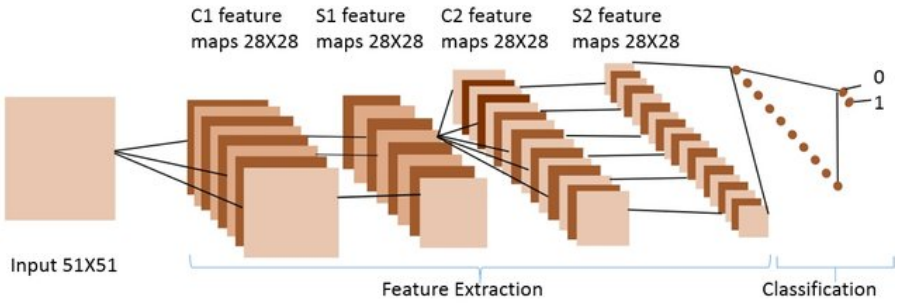



Figure 4: YOLO CNN architecture.

With the introduction of word embeddings [13] and advances in Natural Language Processing, Transformers [14] were developed in 2017, leveraging self-attention mechanisms. Several years later, embeddings were adapted for images, enabling the extraction of information from them, these models titled ViTs (Vision Transformers) [11], were a major milestone, further expanding the capabilities of Computer Vision across the board. One study compared the performance between ViTs and CNNs [15] and found that transformers performed significantly better than state-of-the-art RCNN models (Region-based Convolutional Neural Networks) such as Faster-RCNN [7], Figure 5.

Faster-RCNN, utilizes Region-Based Neural Networks with effective attention mechanisms further improving accuracy, however, these models are often more demanding in terms of computational performance. YOLO

models compensate for the lack of performance and are built to perform real-time object detection tasks suitable for this challenge. Detr, ViT and Co-Detr, are all transformer-based models utilizing Encoder/decoder architectures which is a more effective method in attention mechanisms.

| Model | Type | Backbone | params | $\mathbf{mAP_{test}}$ | $\mathbf{mAP_{val}}$ |
|---|---|---|---|---|---|
| Faster-RCNN [7] | RCNN | VGG-16 [16] | 2.4M | 21.2 | 21.5 |
| Resnet-101 [8] | CNN | Resnet | 44.5M | - | 27.2 |
| YOLOv4 [17] | CNN | CSPDarknet53 [18] | 27.6M | 43.5 | - |
| YOLOv7 [19] | CNN | E-ELAN [20] | 36.9M | 51.2 | 51.4 |
| YOLOv8-m [21] | CNN | CSPDarknet53 | 25.9M | - | 50.2 |
| YOLOv12 [22] | CNN | R-ELAN [22] | 20.2M | - | 52.5 |
| Detr [10] | Transformers | Resnet 50/101 [8] | 60M | 44.9 | - |
| ViT-L [11] | Transformers | Resnet | 361M | - | 49.2 |
| Co-Detr [23] | Transformers | Swin-L [24] | 218M | - | 60.7 |
| Co-Detr | Transformers | ViT-L [11] | 304M | 66.0 | 65.9 |

Table 2: Comparison to the various state-of-the-art models on the COCO dataset

| Model | GFLOPS/FPS | #params | AP | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|---|
| Faster RCNN-DC5 | 320/16 | 166M | 39.0 | 60.5 | 42.3 | 21.4 | 43.5 | 52.5 |
| Faster RCNN-FPN | 180/26 | 42M | 40.2 | 61.0 | 43.8 | 24.2 | 43.5 | 52.0 |
| Faster RCNN-R101-FPN | 246/20 | 60M | 42.0 | 62.5 | 45.9 | 25.2 | 45.6 | 54.6 |
| Faster RCNN-DC5+ | 320/16 | 166M | 41.1 | 61.4 | 44.3 | 22.9 | 45.9 | 55.0 |
| Faster RCNN-FPN+ | 180/26 | 42M | 42.0 | 62.1 | 45.5 | 26.6 | 45.4 | 53.4 |
| Faster RCNN-R101-FPN+ | 246/20 | 60M | 44.0 | 63.9 | **47.8** | **27.2** | 48.1 | 56.0 |
| DETR | 86/28 | 41M | 42.0 | 62.4 | 44.2 | 20.5 | 45.8 | 61.1 |
| DETR-DC5 | 187/12 | 41M | 43.3 | 63.1 | 45.9 | 22.5 | 47.3 | 61.1 |
| DETR-R101 | 152/20 | 60M | 43.5 | 63.8 | 46.4 | 21.9 | 48.0 | 61.8 |
| DETR-DC5-R101 | 253/10 | 60M | **44.9** | **64.7** | 47.7 | 23.7 | **49.5** | **62.3** |

Figure 5: Performance between Faster-RCNN and DETR (Detection Transformer). DC5: Feature Map Dilation (Feature Map Resolution Increase). FPN (Feature Pyramid Networks) (Standard modern CNN architecture). R101 (CNN model with 101 layers)

However, several advantages and disadvantages to utilizing transformers are addressed even further : There are various advantages and disadvantages to utilizing transformers in computer vision tasks and these are clearly addressed in Table 3

| Task | ViTs | CNNs |
|---|---|---|
| **Performance on large datasets** | ✅ Better as dataset size increases | ❌ Worse on large datasets |
| **Performance on small datasets** | ❌ Worse performance on small datasets | ✅ Better on small datasets |
| **Attention map interpretability** | ✅ Easier to visualize attention maps | ❌ Harder to interpret or visualize attention |
| **Pre-trained model availability** | ✅ Can leverage pre-trained checkpoints | ❌ Limited pre-trained options |
| **Model size requirements** | ✅ Can reduce model size significantly | ❌ Larger models needed for same tasks |
| **Multi-modal inputs** | ✅ Easily extendable to multiple input types | ❌ Less flexible for multi-modal inputs |
| **Training time to convergence** | ❌ Requires more training time | ✅ Faster convergence |
| **Local feature extraction** | ❌ Worse at extracting local features | ✅ Specialized in local feature extraction |

Table 3: Comparison of Vision Transformers (ViTs) and CNNs. Table generated using ChatGPT and manually verified

Computer Vision contains four primary tasks that are often used in studies for data retrieval: Classification, Object Detection, Segmentation and Object Tracking

### 2.3.1 Classification

An image is classified into a specific label based on the information depicted in the image. For instance, Figure 6, illustrates an image of a dog and the classification results, the dog was correctly classified as "Old English Sheepdog" (mix of Poodle and Old English Sheepdog)

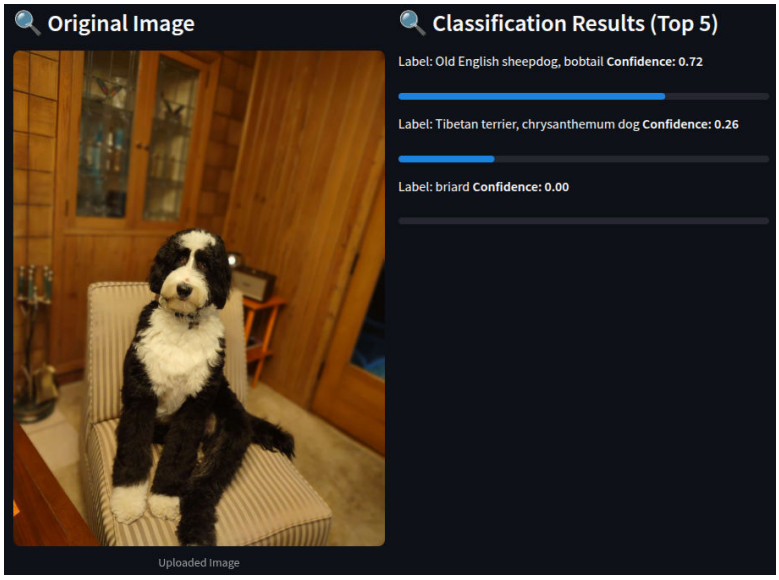



Figure 6: Classification Example

### 2.3.2 Object Detection

Object detection is often used to automate tasks such as detecting multiple objects within images or videos. This task operates by predicting the coordinates of an object, as well as assigning a specific label to them (e.g. person, dog, cat, etc). Figure 7, illustrates the drawn bounding boxes around two objects a dog and a chair. The format of these coordinates is usually provided in two forms (*x, y, w, h*) where '*x*' and '*y*' are the top left coordinates and '*w*' and '*h*' the width and height. Whereas, other models will produce (*x1, y1, x2, y2*) for the top left and bottom right coordinates.


Figure 7: Object Detection

Object detection tasks are often used in combination with image processing techniques to monitor and count the flow of products in conveyor belts. Another implementation is to utilize it to detect defects in tools. Alternatively, instead of image processing another task would be to utilize object tracking that enables long-term monitoring of products, by tracking one or more of the same type of objects in multiple images, this is described in the next section.

### 2.3.3 Object Tracking

Object tracking is a two-step process that always goes hand-in-hand with object detection. The first step is to retrieve the coordinates of the detections and the second step is to pass the list of detections into a tracker. The tracking model is able to monitor each detection and assign a unique identifier to it, simultaneously, the goal of the tracker is to sustain the same identifier for the same object in multiple frames. Figure 8, illustrates this exact process of a detected bird


Figure 8: Object Tracking. Source: [25]

Object Tracking is an excellent task in monitoring long-term data of facilities, such as product flow, machine efficiency and more.

### 2.3.4 Semantic Segmentation

Semantic Segmentation is similar to Object Detection, however, instead of bounding boxes, the corresponding pixels relating to an object in the image are assigned with unique identifiers. Each identifier represents a class describing the detected object. The model will produce polygon coordinates around the object $(x1, y1, x2, y2, ...x_n, y_n)$. Figure 9, illustrates masks overlayed over the object in addition with the labels using a segmentation model.


Figure 9: Segmentation

This technique enables us to extract specific features from individual objects, such as relative object size, coverage percentage and more. Particularly for quality inspection, we can utilize this method to detect defects and calculate their size.

# 3 Existing Datasets

In this section, we expand on various studies that have developed a variety of datasets designed for computer vision tasks particularly in the field of industrial manufacturing.

One study [26] developed a dataset containing images of industrial machine components with visible defects. Furthermore, the goal of the study was to utilize models with various tasks to detect defects on the components using visual inspection equipment. Figure 10, illustrates those tasks with models trained under the BSData dataset, to detect defects on the components.

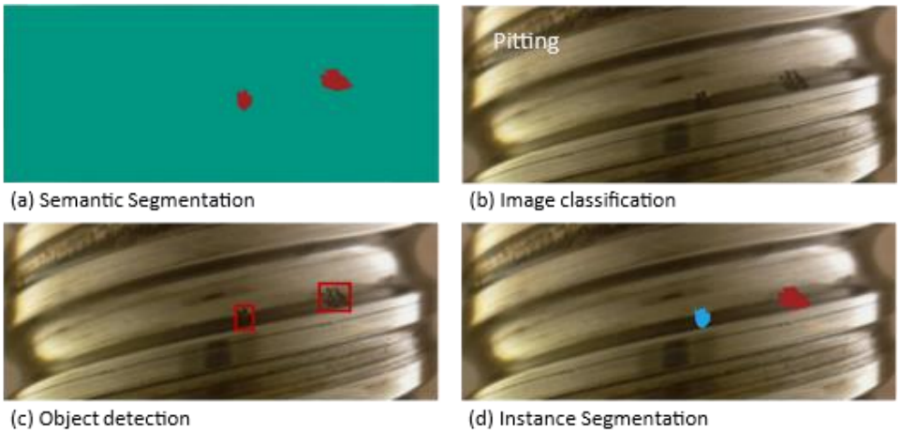



Figure 10: Defect detection of Industrial Machine Tool Components using various computer vision tasks

In regards to quality inspection of paintings, walls and buildings, two more datasets particularly for the segmentation and detection tasks were designed to detect and segment cracks in walls, buildings or paint. The first dataset namely Crack-BPHDR [27] was originally published and developed in Roboflow, the second dataset namely BD3 [28] is a diverse dataset designed specifically for detecting building defects containing up to 3,965 images and with the assistance of augmentation 14,000 images total.

Finally, regarding the detection of objects in conveyor belts transferring items from one point to another, Jocher et. al. [29] developed a dataset aiming to detect medical pills. The dataset is relatively small of about 115 annotated images (train: 92, valid: 23).

# 4 Proposed System IM4VIS: Industrial Manufacturing 4.0 Vision-based Inspection System

This assignment proposes a fully autonomous system designed to monitor incoming tools and equipment to improve the overall efficiency of the production process of the company. The proposed system design illustrates various stages that can be illustrated into real-world scenarios based on the company's demands. Figure 11

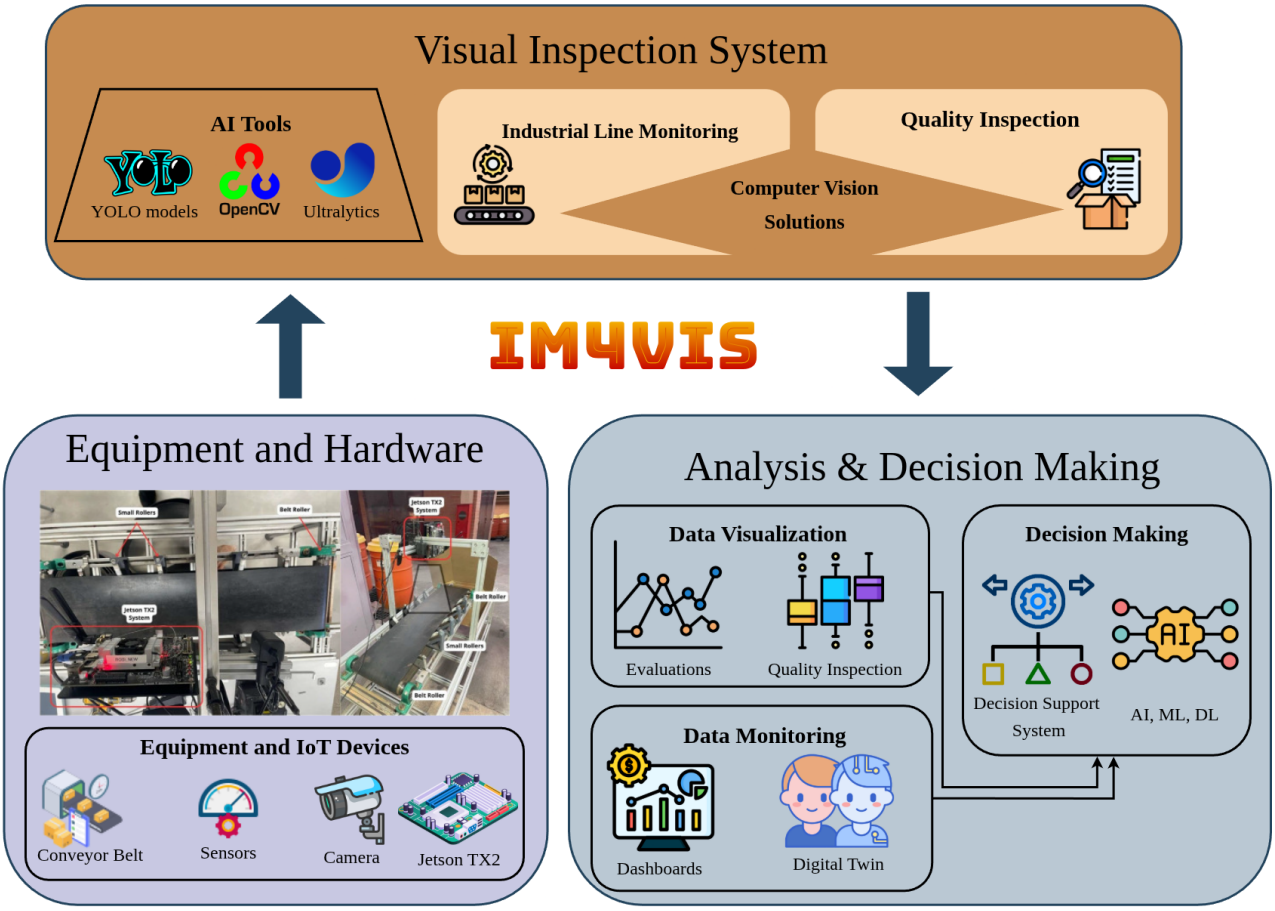



Figure 11: Proposed System Architecture IM4VIS

The system consists of three main layers: (1) Equipment and Hardware, (2) Visual Inspection System and (3) Analysis and Decision-Making. Each layer can effectively be addapted into different fields such as industrial environment. To develop IM4VIS effectively, data acquisition is essential. Although real-world data such as the production line is ideal for this study, it is often challenging and would require the retrieval of data long-term. To address this, for the given challenge we utilized publicly available datasets that offer a practical alternative to the company's demands both for training and validating the system across various tasks. These datasets demonstrate an early benchmark for the SOTA IM4VIS system.

## 4.1 Data Acquisition

High-resolution images and videos can be captured and later produced using static cameras or AR (Augmented Reality) glasses. Using this equipment, data can be gathered, stored and labeled with the use of various advanced labeling tools for computer vision tasks. In practice, a dataset for object detection tasks is built using labeling tools to assign bounding box coordinates in images describing the exact location of an object and a label describing it. There are various labeling tools to build custom datasets for computer vision tasks such as LabelStudio [25], DarkMark [26], Supervisely [27] and finally cvat [28] being one of the more advanced tools available.

In our research, we trained our own selection of segmentation and detection models on the BSData and MedicalPills datasets to evaluate and compare them 4. Furthermore, our study's proposed system architecture IM4VIS, utilizes these datasets to implement two major solutions and facilitate the correct flow of incoming products that are processed by the company.

| Attribute | Dataset | |
|---|---|---|
| | Medical Pills [29] | BSData [26] |
| **CV Task** | Object Detection | Segmentation |
| **Total Images** | 115 | 394 |
| **Training Images** | 92 | 294 |
| **Validation Images** | 23 | 50 |
| **Test Images** | 0 | 50 |

Table 4: Summary of the datasets used for object detection and segmentation tasks.

## 4.2 IM4VIS Architecture

This section expands on each layer from both theoretical and practical perspectives.

### Equipment and Hardware

This layer consists of the physical infrastructure responsible for data acquisition. This layer includes various equipment for data acquisition and is often the initial step in IoT-heavy inspired systems. The technology utilized in this study in this layer is primarily theoretical, but can be adapted into real-world use. Consequently, the equipment utilized in this layer includes from a variety of equipment as stated below:

- **Conveyor Belts**: Often used to automate transportation from one point to another. This study assumes the use of conveyor belts.
- **Sensors**: Many of the sensors used to acquire data either for computer vision or environmental data usually revolve around cameras or LIDAR (Light detection and Ranging). Several other sensors include gas-detection, accelerometers, thermal imagery and more. Additionally, mobile phones can be utilized as an effective method to acquire visual data. Our study mostly utilizes vision, therefore, the use of cameras is proprietary.
- **IoT Devices**: To process data derived from the sensors in real-time hardware equipped with CUDA technology is utilized to transfer processed data across the web. This is also called Edge-Computing and is an important layer in IoT-driven systems. To train, evaluate and test our computer vision models, we utilized a heavy computer equipped with a RTX 4090. However, in real-world cases, the use of lower performance hardware is utilized such as Jetson TX2.

It includes equipment (conveyor belts), IoT devices such as cameras, sensors, NVIDIA Jetson modules and CUDA-based computers [30], [31] for accelerated image processing.

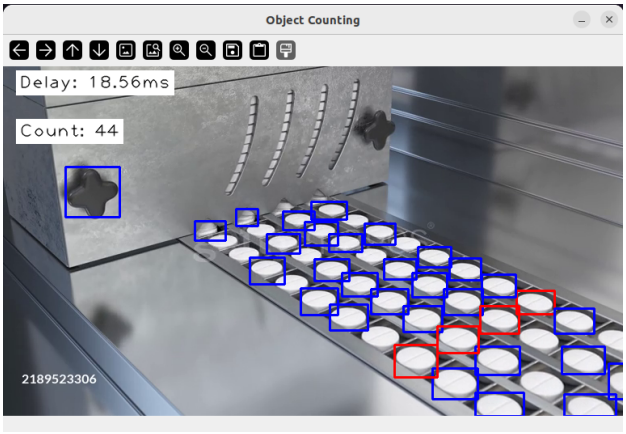
### Visual Inspection System

This layer is the software-side of the system that processes the images acquired from the hardware layer. It includes two main solutions utilizing computer vision methods: **Industrial Line Monitoring** and **Quality Inspection**.

**Industrial Line Monitoring**:

In this system we trained several models in the YOLO family, evaluated them and utilized the best performing model in terms of accuracy per-task. The model we utilized for this system is YOLOv8x that aims to detect medical pills in conveyor belts. This subsystem utilizes detection and tracking techniques to count medical pills in conveyor belts. The counting methodology works by drawing a region of interest in the image as illustrated in Figure 13a and pressing '$s$' to submit the region, otherwise, '$q$' to close the program. Once the region is submitted, the entire video is processed and the pills are detected using the trained yolo model and tracked using the BotSort [32] tracking algorithm. Simultaneously, any tracked pill that passes into the region is counted once as also illustrated in Figure 12b


(a) Counting Region Initialization

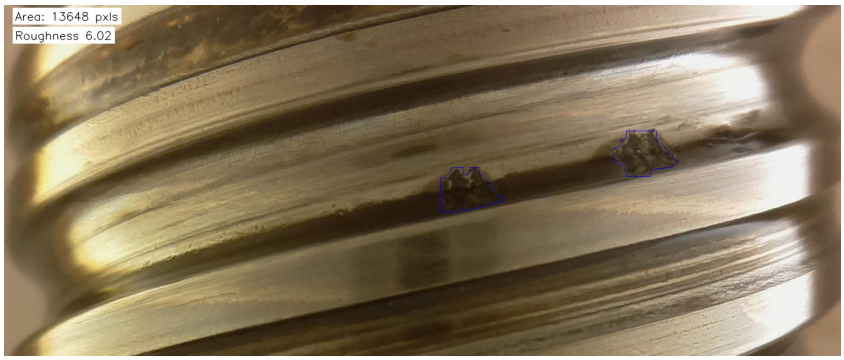



(b) Industrial Line Monitoring Inference

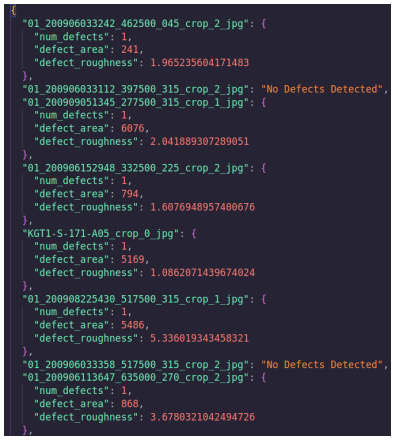
Figure 12: Demo example of the Industrial Line Monitoring system

**Quality Inspection**:

The quality inspection system utilizes a trained segmentation model, YOLOv12X trained under the BSData dataset with the goal of detecting defects in *incoming products and equipment* for the company. This method effectively detects defects and extracts their location in the image, additionally, with standard image processing techniques, we can calculate the defect size (in pixels) and surface roughness. Each image is processed individually and every information is extracted into a json file.




(a) Defect Detection




(b) Defect Data Extraction

Figure 13: Demo example of the Industrial Line Monitoring system

## Analysis & Decision-Making

The system's general intelligence layer, aims to analyze data from the previous stages. This layer can provide a variety of solutions to optimize the company's production efficiency:

- Data Visualization (Implement)
- Digital Twin
- Dashboard
- System Evaluations (Implemented)
- Decision Making

**System Evaluation**:

To observe the performance of the previous layer of the system (Visual Inspection System), we evaluated each model individually for each dataset and task utilized in the study. The segmentation and detection models were evaluated using the standard MSCOCO [33] evaluation tool. Figure 5, illustrates the evaluation results for various YOLO models for the segmentation and detection tasks under the datasets BSData and MedicalPills.

| Model | Params (M) | GFLOPs | **BSData** [26] (Segmentation) | | **Medical-Pills** [29] (Detection) | |
|---|---|---|---|---|---|---|
| | | | $\mathbf{mAP_{50}}$ | $\mathbf{mAP_{50:95}}$ | $\mathbf{mAP_{50}}$ | $\mathbf{mAP_{50:95}}$ |
| YOLOv8m [21] | 27.22/25.84 | 110.0/78.7 | 0.751 | 0.483 | 0.992 | 0.819 |
| YOLOv8l | 45.91/43.60 | 220.1/164.8 | 0.744 | 0.486 | 0.993 | 0.830 |
| YOLOv8x | 71.72/68.12 | 343.7/257.4 | 0.705 | 0.453 | 0.993 | 0.832 |
| YOLOv11m [34] | 22.33/20.03 | 123.0/67.6 | 0.757 | 0.467 | 0.992 | 0.813 |
| YOLOv11l | 27.58/25.28 | 141.9/86.6 | 0.729 | 0.453 | 0.993 | 0.829 |
| YOLOv11x | 62.00/56.82 | 318.5/194.4 | 0.713 | 0.457 | 0.993 | 0.827 |
| YOLOv12m [22] | 21.93/20.10 | 114.9/67.1 | 0.791 | 0.503 | 0.993 | 0.828 |
| YOLOv12l | 28.16/26.33 | 136.3/88.5 | 0.776 | 0.475 | 0.993 | 0.826 |
| YOLOv12x | 63.14/59.04 | 305.6/198.5 | 0.801 | 0.511 | 0.993 | 0.827 |

Table 5: Evaluation of YOLOv8, YOLOv11, and YOLOv12 segmentation and detection models on BSData and Medical-Pills datasets. Green indicates best performance and red the least best performance in each column respectively. Training Parameters: (a) $BatchSize = 0.8$ (b) $Epochs = 80$

Training under the MedicalPills dataset was very challenging, due to its overall size of only 115 images. Larger models are significantly more prone to overfitting especially in small datasets. Overfitting implies that the model was over-trained on the training dataset resulting in higher accuracy compared to a testing dataset. This behavior is also observed in the previous evaluation results, Table 5, particularly for the YOLOv8x that is the largest model and was evaluated with the highest mAP. To address this issue, we performed several experiments for this analysis on the Medical-Pills dataset with several parameters.

**Experiment 1:** Batch Utilization: 80%, Epochs: 80

**Experiment 2:** Batch Utilization: 95%, Epochs: 50

**Experiment 3:** Batch Utilization: 95%, Epochs: 30

After training and evaluating each model individually for each experiment, we observed the following results (Table 6). YOLOv8x performs best in most cases compared to other models.

| Model | Experiment 1 | | Experiment 2 | | Experiment 3 | |
|---|---|---|---|---|---|---|
| | $\mathbf{mAP_{50}}$ | $\mathbf{mAP_{50-95}}$ | $\mathbf{mAP_{50}}$ | $\mathbf{mAP_{50-95}}$ | $\mathbf{mAP_{50}}$ | $\mathbf{mAP_{50-95}}$ |
| yolov8m | 0.992 | 0.819 | 0.993 | 0.8167 | 0.992 | 0.8044 |
| yolov8l | 0.993 | 0.830 | 0.992 | 0.8248 | 0.993 | 0.81128 |
| yolov8x | 0.993 | 0.832 | 0.993 | 0.8292 | 0.992 | 0.8183 |
| yolo11m | 0.992 | 0.813 | 0.993 | 0.8175 | 0.992 | 0.7984 |
| yolo11l | 0.993 | 0.829 | 0.993 | 0.8192 | 0.992 | 0.8030 |
| yolo11x | 0.993 | 0.827 | 0.992 | 0.8174 | 0.993 | 0.8156 |
| yolo12m | 0.992 | 0.828 | 0.993 | 0.8134 | 0.992 | 0.8097 |
| yolo12l | 0.993 | 0.826 | 0.993 | 0.8234 | 0.993 | 0.8153 |
| yolo12x | 0.992 | 0.827 | 0.992 | 0.8221 | 0.992 | 0.8112 |

Table 6: Experiment Evaluation Results under the dataset MedicalPills [29]

In both experiments 1 and 2 the models overfitted. Alternatively, experiment 3 did not overfit with the least amount of epochs. While the results are not visible in Table 6, we produced a demo video (Click Me To Redirect to Demo Video) with a side-by-side comparison of yolov8x inference along with the tracking algorithm in both experiments 1 (left) and 3 (right).

## 5 Conclusion

This study introduced a state-of-the-art system with the goal of improving industrial manufacturing using vision inspection systems. The proposed model namely IM4VIS (Industrial Manufacturing 4.0 Vision Inspection System), aimed to improve the production efficiency of the company using computer vision tools. Furthermore, several solutions are introduced in this study: 1) fine-tuned detection model YOLOv8x to achieve high accuracy detection 2) utilization of a tracking model that enables us to count objects in specific regions in a video. 3) quality inspection by detecting defects on tools and incoming equipment 4) defect feature extraction to analyze the severity of the defects following with data such as number of defects, defect size and defect surface roughness. Consequently, this study performed various evaluations under several computer vision models. The best performing computer vision models for the segmentation task was YOLOv12x, where as the best performing model for the detection task was yolov8x. This indicates the importance in a dataset overall quality and quantity. Particularly for the BSData dataset, YOLOv12x performed better with 0.058 (5.8%) higher mAP, where as for the Medical Pills dataset, YOLOv12x performed slightly worse than YOLOv8x with 0.005 0.5% less mAP.

## 6 Future Work

Although, this system design demonstrates a significant contribution to the literature, the design is still lacking a significant portion since several stages are employed theoretically. To achieve a fully autonomous, end-to-end, and sustainable computer vision system, constructing a custom dataset is essential to replace public datasets and address real-world problems. Future development should focus on deployign physical IoT sensors and micro-controller systems (e.g. Jetson TX2) for real-time data retrieval and processing from the productionline. Furthermore, the implementation of a digital twin and a real time dashboard will enable engineers and manufacturers to remotely monitor the system's performance long-term. Additionally, components that can enhance IM4VIS include decision-making logic for proactive defect detection. These methods will address many of the company's demands by improving the production efficiency long-term.

## References

[1] A. Luis Bustamante, M. A. Patricio, and J. M. Molina, "Thinger.io: An open source platform for deploying data fusion applications in iot environments," *Sensors*, vol. 19, no. 5, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/5/1044

[2] S. K. Das, M. H. Uddin, and S. Baidya, "Edge-assisted collaborative digital twin for safety-critical robotics in industrial iot," *arXiv preprint arXiv:2209.12854*, 2022.

[3] S. Islam, M. S. Tanvir, M. R. Habib, T. T. Shawmee, M. A. Ahmed, T. Ferdous, M. R. Arefin, and S. Alam, "Autonomous driving vehicle system using lidar sensor," in *Intelligent Data Communication Technologies and Internet of Things*, D. J. Hemanth, D. Pelusi, and C. Vuppalapati, Eds. Singapore: Springer Nature Singapore, 2022, pp. 345–358.

[4] D. Hercog, T. Lerher, M. Truntič, and O. Težak, "Design and implementation of esp32-based iot devices," *Sensors*, vol. 23, no. 15, p. 6739, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/15/6739

[5] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[10] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*. Springer, 2020, pp. 213–229.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[15] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*. Springer, 2020, pp. 213–229.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[18] C. Wang, H. Liao, I. Yeh, Y. Wu, P. Chen, and J. Hsieh, "Cspnet: A new backbone that can enhance learning capability of cnn corr," *vol. abs/1911.11929*, 2019.

[19] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 7464–7475.

[20] C.-Y. Wang, H.-Y. M. Liao, and I.-H. Yeh, "Designing network design strategies through gradient path analysis," *arXiv preprint arXiv:2211.04800*, 2022.

[21] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolov8," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[22] Y. Tian, Q. Ye, and D. Doermann, "Yolov12: Attention-centric real-time object detectors," *arXiv preprint arXiv:2502.12524*, 2025.

[23] Z. Zong, G. Song, and Y. Liu, "Detrs with collaborative hybrid assignments training," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023, pp. 6748–6758.

[24] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[25] D. Mpouziotas, P. Karvelis, and C. Stylios, "Advanced computer vision methods for tracking wild birds from drone footage," *Drones*, vol. 8, no. 6, p. 259, 2024.

[26] T. Schlagenhauf and M. Landwehr, "Industrial machine tool component surface defect dataset," *Data in Brief*, vol. 39, p. 107643, 2021.

[27] University, "crack dataset," dec 2022, visited on 2024-01-23. [Online]. Available: https://universe.roboflow.com/university-bswxt/crack-bphdr

[28] P. Kottari and P. Arjunan, "Bd3: Building defects detection dataset for benchmarking computer vision techniques for automated defect identification," in *Proceedings of the 11th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, 2024, pp. 297–301.

[29] G. Jocher and M. Rizwan, "Ultralytics datasets: Medical-pills detection dataset," Dec 2024. [Online]. Available: https://docs.ultralytics.com/datasets/detect/medical-pills/

[30] D. Kirk, "Nvidia cuda software and gpu parallel computing architecture," in *6th International Symposium on Memory Management*, vol. 7, 10 21st of October, 2007, pp. 103–104, (Accessed on Feb 19, 2024).

[31] R. Yi, T. Cao, A. Zhou, X. Ma, S. Wang, and M. Xu, "Boosting dnn cold inference on edge devices," *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*, 15th of June, 2022, (Accessed on Apr 5, 2024). [Online]. Available: https://api.semanticscholar.org/CorpusID:249674551

[32] N. Aharon, R. Orfaig, and B.-Z. Bobrovsky, "Bot-sort: Robust associations multi-pedestrian tracking," *arXiv preprint arXiv:2206.14651*, 2022.

[33] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part v 13*. Springer, 2014, pp. 740–755.

[34] R. Khanam and M. Hussain, "Yolov11: An overview of the key architectural enhancements," *arXiv preprint arXiv:2410.17725*, 2024.

# Appendix

## A Training, Evaluation, Prediction

### A.1 python code: train_on_BSData.py

```python
from ultralytics import YOLO
import torch

device = "cuda" if torch.cuda.is_available() else "cpu"
models_list = ['yolov8m-seg.pt', 'yolov8l-seg.pt', 'yolov8x-seg.pt',
               'yolo11m-seg.pt', 'yolo11l-seg.pt', 'yolo11x-seg.pt',
               'yolov12m-seg.pt', 'yolov12l-seg.pt', 'yolov12x-seg.pt']

for model_name in models_list:
        model = YOLO(model_name)
        model_name = model_name.split('.')[0]
        model.train(
                data='/home/mpouziotasd/Documents/Personal/industry 4.0/Visual Inspection
                ↪  System/datasets/BSData/data.yaml',
                epochs=80,
                batch=0.8,
                name=f'{model_name}-BSData'
                )
```

### A.2 python code: train_on_MedicalPills.py

```python
from ultralytics import YOLO

models_list = ['yolov8m.pt', 'yolov8l.pt', 'yolov8x.pt',
               'yolo11m.pt', 'yolo11l.pt', 'yolo11x.pt',
               'yolo12m.pt', 'yolo12l.pt', 'yolo12x.pt']
```

```

for model_name in models_list:
    model = YOLO(model_name)
    model_name = model_name.split('.')[0]
    results = model.train(data="medical-pills.yaml", epochs=30, imgsz=640, name=f'{model_name}-MedicalPills',
    ↪  batch=0.95)
```

## A.3 python code: train_custom.py

Previously made code...

```
import argparse
from ultralytics import YOLO

def begin_train(cfg, data, epochs, imgsz, batch, name, verbose=True):
    model = YOLO(cfg)
    model.train(
        data=data,
        epochs=epochs,
        imgsz=imgsz,
        batch=batch,
        name=name
    )

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Train a YOLOv8 model.")
    parser.add_argument('--cfg', type=str, required=True, help="Path to the YOLOv8 configuration file (e.g.,
    ↪  yolov8-M.yaml).")
    parser.add_argument('--data', type=str, default="F:\Transfer\Courses - Masters PMS UoII_New2\Industry
    ↪  4.0/assignment\computer vision\datasets\BSData\data.yaml", help="Path to the dataset YAML file.")
    parser.add_argument('--epochs', type=int, default=150, help="Number of training epochs.")
    parser.add_argument('--imgsz', type=int, default=640, help="Image size for training.")
    parser.add_argument('--batch', type=int, default=16, help="Batch size for training.")
    parser.add_argument('--name', type=str, default='MyTrain', help="Name of the training session.")

    args = parser.parse_args()
    begin_train(
        cfg=args.cfg,
        data=args.data,
        epochs=args.epochs,
        imgsz=args.imgsz,
        batch=args.batch,
        name=args.name
    )
```

## A.4 python code: prediction.py

Predicts on all the BSData images. Can also be revised for Medical Pills dataset.

```
from ultralytics import YOLO
import os

model_path = "models/yolov8m-seg.pt"  # Path to the pre-trained YOLOv8 model
test_path = "../../datasets/BSData/test/images"
out_dir = "predict_outputs"
if not os.path.exists(out_dir):
    os.makedirs(out_dir)

images = os.listdir(test_path)

def load_model(model_path):
    model = YOLO(model_path)
    return model

model = load_model(model_path)


for image in images:
    img_src = f"{test_path}/{image}"
    results = model(img_src, save=True, project=out_dir, exist_ok=True)[0]
```

# B Visual Inspection System

## B.1 python code: object_counting_noGUI.py

Attempted to make a GUI-based system but backed down due to several issues with website and GUI interactions...

```python
import argparse
import os
import cv2
import numpy as np
import time
from ultralytics import solutions
from utils.model_utilities import load_model, detect
from utils.img_utils import draw_text, draw_data, draw_counting_region

def main():
    # Code refined using ChatGPT...
    parser = argparse.ArgumentParser(description="Object Counting System")
    parser.add_argument("--video", required=True,
                        help="Path to input video file")
    parser.add_argument("--model", required=True,
                        help="Model name (without .pt extension)")
    parser.add_argument("--output",
                        help="Path to output video file (optional)")
    parser.add_argument("--classes", type=int, nargs='+', default=[0],
                        help="List of class IDs to detect (default: 0)")

    args = parser.parse_args()

    cap = None
    model = None
    counter = None
    region_points = []
    prev_time = time.time()
    writer = None
    print(args.output)
    try:
        cap = cv2.VideoCapture(args.video)


        if not cap.isOpened():
            raise ValueError(f"Could not open video: {args.video}")

        if not os.path.exists(args.model):
            raise FileNotFoundError(f"Model not found: {args.model}")
        model = load_model(args.model)[0]

        ret, frame = cap.read()
        if not ret:
            raise RuntimeError("Could not read first frame from video")

        region_points, _ = draw_counting_region(frame)
        if region_points is None:
            print("Region selection canceled. Exiting.")
            return

        counter = solutions.ObjectCounter(
            region=region_points,
            model=args.model,
            classes=args.classes,
            show=False,
            show_conf=False,
            show_labels=False,
            verbose=False,
            show_in=False,
            show_out=False,
            line_width=2
        )

```

```
        if args.output:
            fps = cap.get(cv2.CAP_PROP_FPS)
            width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
            height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
            fourcc = cv2.VideoWriter_fourcc(*'mp4v')
            writer = cv2.VideoWriter(args.output, fourcc, fps, (width, height))

        print("Starting object counting. Press 'q' to exit...")
        while True:
            ret, frame = cap.read()
            if not ret:
                break

            current_time = time.time()
            delta = current_time - prev_time
            prev_time = current_time
            delay_ms = delta * 1000

            results = detect(model, frame, device="0")[0]

            counter_results = counter(frame.copy())
            print(counter_results, type(counter_results))

            bboxes = results.boxes.xyxy
            clss = results.boxes.cls
            if bboxes is not None:
                processed_frame = draw_data(frame, bboxes, clss, region_points)
                draw_text(frame, text=f"Delay: {delay_ms:.2f}ms")
                try:
                    draw_text(frame, text=f'Count: {counter_results.in_count}', position=(20, 80))
                except:
                    draw_text(frame, text=f'Count: 0', position=(20, 50))

            else:
                processed_frame = frame

            if writer:
                writer.write(processed_frame)
                print("WRITING...")
            else:
                cv2.imshow("Object Counting", processed_frame)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break

    except Exception as e:
        print(f"Error: {str(e)}")
    finally:
        if cap:
            cap.release()
        if writer:
            writer.release()
        cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

## B.2 python code: quality_inspection.py

```
from ultralytics import YOLO
import numpy as np
import cv2 as cv
import os
from utils.img_utils import draw_polygons, get_mask_area, calculate_roughness, extract_mask, draw_text

input_dir = "data/defects/"

out_dir = "results"

if not os.path.exists(out_dir):
    os.mkdir(out_dir)
```

```

if not os.path.exists(f"{out_dir}/imgs/"):
    os.mkdir(f"{out_dir}/imgs/")

model = YOLO("models/yolov12x-BSData.pt")
imgs_src = os.listdir(input_dir)

def process_image(img):
    results = model(img)[0]
    annotated_img = img.copy()

    inspection_info = {}
    if results.masks:
        masks = results.masks.xy
        num_masks = 0
        roughness_sum = 0
        area_sum = 0
        num_masks = len(masks)
        for i, mask in enumerate(masks):
            mask = np.array(mask, np.int32)
            annotated_img = draw_polygons(img, mask)
            area = get_mask_area(mask, img.shape)
            defect_crop = extract_mask(img, mask)
            roughness = calculate_roughness(defect_crop)

            roughness_sum += roughness
            area_sum += area
        draw_text(annotated_img, f"Area: {area_sum} pxls")
        draw_text(annotated_img, f"Roughness {roughness_sum:.2f}", position=(20, 55))
        inspection_info = {
                    'num_defects': num_masks,
                    'defect_area': area_sum,
                    'defect_roughness': roughness_sum
        }
    else:
        inspection_info = "No Defects Detected"
    return annotated_img, inspection_info

detection_info = {}

for it, img_src in enumerate(imgs_src):
    file_name = img_src.split('.')[0]
    print(f"Processing Image: [{img_src}]")
    img_path = f"{input_dir}{img_src}"
    img_cv = cv.imread(img_path)
    annotated_img, inspection_info = process_image(img_cv)
    detection_info[file_name] = inspection_info
    out_img_path = f"results/imgs/{img_src}"

    cv.imwrite(out_img_path, annotated_img)

import json
with open(f"{out_dir}/results.json", 'w') as f:
    json.dump(detection_info, f, indent=2)
```

# C Utilities

## C.1 python code: img_utils.py

```
import cv2 as cv
import numpy as np

color_ranges = {
    "red": (0, 0, 255),
    "green": (0, 255, 0),
    "blue": (255, 0, 0),
    "yellow": (0, 255, 255),
    "white": (255, 255, 255)
}
```

```python

def draw_text(frame, text, position=(20, 20), background=True, background_color=(255, 255, 255), text_color=(0, 0, 0)):
    x, y = int(position[0]), int(position[1])
    font_scale = 0.7
    thickness = 1
    padding = 4

    (text_w, text_h), baseline = cv.getTextSize(text, cv.FONT_HERSHEY_SIMPLEX, font_scale, thickness)
    text_w, text_h = int(text_w), int(text_h + baseline)

    text_x = x
    text_y = y + text_h // 2 - baseline // 2
    if background:
        bg_top = y - text_h // 2 - padding
        bg_bottom = y + text_h // 2 + padding
        bg_left = x - padding
        bg_right = x + text_w + padding
        cv.rectangle(frame,
                     (bg_left, bg_top),
                     (bg_right, bg_bottom),
                     background_color, -1)

    cv.putText(
        frame,
        text,
        (text_x, text_y),
        cv.FONT_HERSHEY_SIMPLEX,
        fontScale=font_scale,
        color=text_color,
        thickness=thickness,
        lineType=cv.LINE_AA
    )

def is_in_boundaries(obj_point, region_points):
    obj_point = np.array(obj_point, dtype=np.float32)
    region_points = np.array(region_points, dtype=np.float32)
    if region_points.ndim == 2:
        region_points = region_points.reshape((-1, 1, 2))
    return cv.pointPolygonTest(region_points, tuple(obj_point), False) >= 0

def draw_data(frame, bboxes, clss, region_points):
    for bbox, _cls in zip(bboxes, clss):
        x1, y1, x2, y2 = map(int, bbox)
        col = color_ranges['red'] if is_in_boundaries(((x1+x2)//2, (y1+y2)//2), region_points) else color_ranges['blue']
        cv.rectangle(frame, (x1, y1), (x2, y2), col, 2)

    return frame

def draw_polygons(frame, pts):
    return cv.polylines(frame, [pts], True, (255, 0, 0), 1)

def get_mask_area(mask, shape):
    binary_mask = np.zeros(shape[:2], dtype=np.uint8)
    cv.fillPoly(binary_mask, [mask], 1)
    return int(np.sum(binary_mask))

def extract_mask(image, mask):
    mask_img = np.zeros(image.shape[:2], dtype=np.uint8)
    cv.fillPoly(mask_img, [mask], 255)
    masked = cv.bitwise_and(image, image, mask=mask_img)

    x, y, w, h = cv.boundingRect(mask)
    cropped = masked[y:y+h, x:x+w]
    return cropped

def calculate_roughness(mask):
    edges = cv.Canny(mask, threshold1=10, threshold2=30)

```

```
    contours, _ = cv.findContours(edges, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
    if not contours:
        return 0.0

    cnt = max(contours, key=cv.contourArea)
    peri = cv.arcLength(cnt, True)
    hull = cv.convexHull(cnt)
    hull_peri = cv.arcLength(hull, True)
    if hull_peri == 0:
        return 0.0
    return peri / hull_peri


def draw_counting_region(frame):
    """
        Description:
            Defines an interactable UI on an image to draw a rectangle region of interest using points for the
            ↪  counting model.
            Click on one point to another and form a rectangle. Minimum of 3 points are required.
        Interactions:
            i) 'Mouse Click': Click to a certain location to define a point for the rectangle.
            ii) 'S': Press 'S' to submit the region of interest onto the tracking model.
            iii) 'Q': Press 'Q' to exit the window and the program.
    """

    working_frame = frame.copy()
    original_frame = frame.copy()
    points = []
    submitted = False
    window_name = "Draw Counting Region"
    box = None

    def mouse_callback(event, x, y, flags, param):
        nonlocal working_frame, points, submitted

        if submitted:
            return

        if event == cv.EVENT_LBUTTONDOWN:
            points.append((x, y))

        elif event == cv.EVENT_RBUTTONDOWN and points:
            points.pop()

        working_frame = original_frame.copy()
        for p in points:
            cv.circle(working_frame, p, 5, (0, 255, 0), -1)

        if len(points) >= 3:
            rect = cv.minAreaRect(np.array(points))
            box_local = cv.boxPoints(rect)
            box_local = np.intp(box_local)
            cv.drawContours(working_frame, [box_local], 0, (0, 0, 255), 2)

        cv.imshow(window_name, working_frame)

    cv.namedWindow(window_name)
    cv.setMouseCallback(window_name, mouse_callback)
    cv.imshow(window_name, working_frame)

    while True:
        key = cv.waitKey(1) & 0xFF
        if key == ord('s') and len(points) >= 3:
            rect = cv.minAreaRect(np.array(points))
            box = cv.boxPoints(rect)
            box = np.intp(box)
            submitted = True
            break
        elif key == ord('q') or key == 27:
            break
```

```
148
149     cv.destroyWindow(window_name)
150     cv.waitKey(1)
151
152     if submitted:
153         result_frame = original_frame.copy()
154         cv.drawContours(result_frame, [box], 0, (0, 255, 0), 2)
155         for p in points:
156             cv.circle(result_frame, p, 5, (0, 255, 0), -1)
157         return box, result_frame
158     else:
159         return None, original_frame
```

## C.2 python code: model_utilities.py

Optional, not utilized...

```
1  from ultralytics import YOLO
2
3
4  def load_model(model_path=None):
5      """
6          Description:
7          Returns the loaded Computer Vision model using Ultralytics
8
9          Returns:
10             model: YOLO
11             status: int
12     """
13     if not model_path:
14         print("Warning: Model path not set")
15         return None, -2
16
17     try:
18         model = YOLO(model_path)
19         # model.set_classes(['Storage Box'])
20         return model, 0
21     except Exception as e:
22         print("Error when loading the model", e)
23         return None, -1
24
25 def detect(model, frame, device='cpu'):
26     """
27         Inference using CPU or GPU
28         Returns detections as an Ultraytics object using the loaded model.
29     """
30     results = model(frame, device=device)
31     return results
32
```