chapter [0pt] Chapter :

Alexandria University
Faculty of Engineering
Communication and Electronics Department

Graduation Project EEC 401 - 402

Fall-2023 & Spring -2024

Controlling the Performance in a Developed Large Institute

Authors

Eng. Aalaa Yousry Bahi El-din
Eng. Toqa Ayman Gomaa Ezzatly
Eng. Habiba Samir Abd El-Hafiz
Eng. Reem Sabry Mohammed El-Refaay
Eng. Nada El-sayed Mohammed Ali
Eng. Rewan Khaled Abd El-Kareem

Supervisor: Prof. Hassan El-kamshoshy

Table of Contents

List of Figures

Dedication

We dedicate this work to the collective spirit of resilience, determination, and collaboration that fueled our journey through the challenges of academia and the creation of this project.

To our families and friends, whose unwavering support and understanding have been the pillars of strength throughout this endeavor. Your encouragement and belief in our abilities have been the driving force behind our success.

To our supervisor and educators, who have imparted knowledge, guidance, and invaluable insights that shaped our understanding and fueled our passion for learning.

To each member of our team, whose unique talents, diverse perspectives, and shared commitment have turned this vision into reality. Your dedication and hard work have left an indelible mark on this project.

This achievement is not just a culmination of academic pursuits but a celebration of the bonds forged, the lessons learned, and the collective triumph of a journey that we embarked on together.

As we stand on the brink of new beginnings, may this dedication be a testament to the shared dreams, hard work, and the enduring friendships that have defined our academic adventure.

With gratitude and pride,

Authors .

Acknowledgment

We express our deepest gratitude to all who contributed to this project. The completion of this endeavor was only possible with the support, guidance, and encouragement from various individuals and institutions.

Special thanks to our project advisor Prof. Hassan El-kamshoshy for invaluable expertise and mentorship. Your feedback and encouragement propelled us forward, and we appreciate the knowledge and insights you shared.

Our appreciation goes to the faculty members of communication and electronics Department for shaping our academic journey.

Thanks to our families and friends for their unwavering support. Your belief in our abilities provided the emotional sustenance needed to navigate this project.

To our fellow students and peers, thank you for the camaraderie and collaborative efforts that enriched our learning experience.

Lastly, we appreciate the collective effort of our team members. Your hard work and dedication made this journey memorable and rewarding.

This project is a testament to the collaborative spirit in academic settings, and we are thankful for all contributions that brought it to fruition.

With gratitude,

Authors.

Abstract

In the rapidly evolving landscape of industrial automation, ensuring real-time performance and safety monitoring is critical for optimizing operations and protecting workers. This project presents a comprehensive monitoring system leveraging the SX1278 LoRa transceiver for long-range, reliable wireless communication, and the You Only Look Once (YOLO) object detection algorithm implemented on an STM32 microcontroller. The system integrates various sensors (temperature, humidity, vibration, gas) to monitor environmental and operational parameters, transmitting data via the SX1278 LoRa modules to a central receiver. The STM32 microcontroller processes and aggregates this data in real-time, ensuring minimal latency and high reliability. Concurrently, strategically placed cameras in critical factory areas feed visual data to the YOLO algorithm on the STM32, detecting objects and potential hazards with high accuracy. This dual approach enables robust safety monitoring, generating alerts and initiating predefined actions when hazards are detected. The integration of LoRa for communication and YOLO for visual analysis, combined with edge computing capabilities of the STM32, offers a scalable, efficient solution for enhancing safety and performance in industrial environments. This project underscores the potential of combining advanced communication protocols and real-time image processing algorithms to create intelligent, responsive industrial monitoring systems.

Introduction

In recent years, the industrial landscape has witnessed a profound transformation propelled by the integration of advanced technologies such as Internet of Things (IoT), artificial intelligence (AI), and edge computing. This convergence has ushered in the era of smart factories, where interconnected systems and automated processes are revolutionizing traditional manufacturing paradigms. Our graduation project is situated at the forefront of this technological evolution, aiming to realize the potential of a smart factory through a comprehensive implementation framework.

Central to our project is the utilization of computer vision technologies to enhance operational capabilities. By employing sophisticated algorithms, we enable precise tracking of products throughout the production line, ensuring efficient inventory management and quality control. Simultaneously, our system monitors worker safety by detecting compliance with safety protocols such as wearing helmets, and it employs AI-driven analytics to assess worker health in real-time, detecting signs of fatigue or distress that may compromise safety or productivity.

Moreover, our project integrates IoT systems equipped with advanced sensors like the MQ-4 and DHT11 using the power of STM32F103C8TX, enhancing data acquisition capabilities critical for informed decision-making. These sensors enable continuous monitoring of environmental parameters and process variables, facilitating predictive maintenance strategies to preemptively address equipment failures and optimize production uptime.

At the core of our solution is a sophisticated desktop application developed in Python, which serves as the nerve center for data visualization, analysis, and operational control. This application not only provides real-time insights into sensor readings and computer vision outputs but also incorporates machine learning algorithms to predict anomalies and trigger alerts for timely intervention. Additionally, leveraging LoRa technology, our system ensures robust communication between IoT devices and the central application, enhancing reliability and scalability across the factory floor.

By aligning with rigorous academic principles and leveraging state-of-the-art technologies, our project not only aims to demonstrate technical prowess but also seeks to address critical challenges in industrial automa-

tion. Through this endeavor, we aspire to contribute significantly to the advancement of smart manufacturing, paving the way for more efficient, adaptive, and secure industrial environments in the digital age.

Recent researches in automotive factories:

In recent years, the proliferation of smart factory technologies has been accompanied by compelling statistics underscoring their transformative impact on industrial operations. According to a report by Grand View Research, the global smart factory market size is projected to reach USD 395.24 billion by 2028, growing at a compound annual growth rate (CAGR) of 13.3% from 2021 to 2028. This growth is driven by the increasing adoption of automation and digitalization to enhance production efficiency and reduce operational costs across diverse industries.

Global smart factory market size research overview :

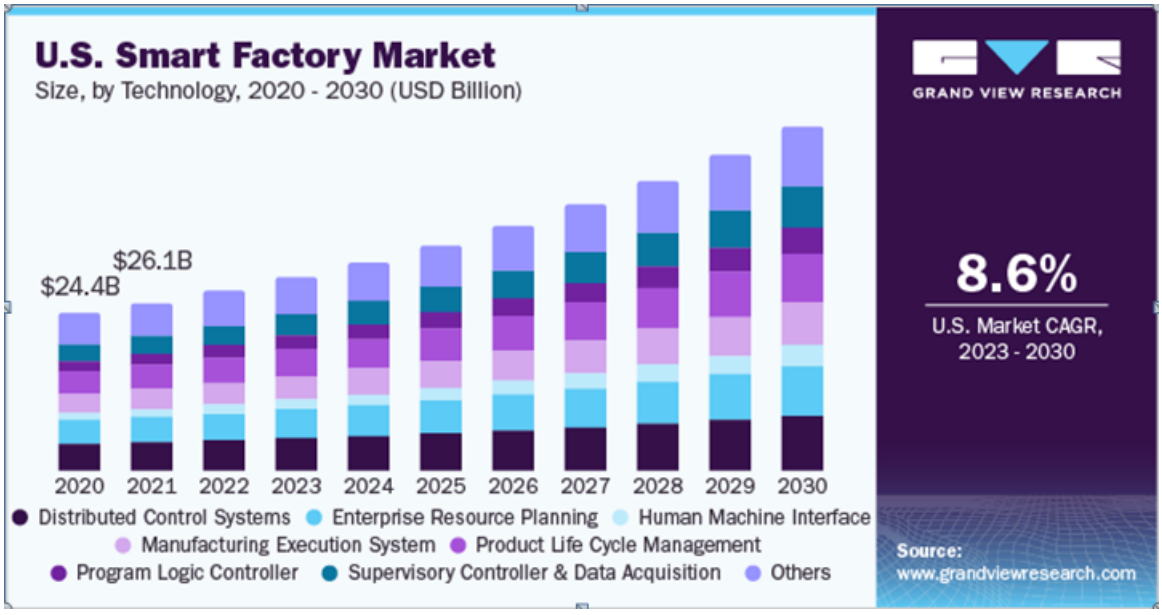

Figure 0.1: Global smart factory market size research overview

The industrial robotics segment accounted for the largest market share of over 41% in 2022. Industrial robots are increasingly preferred by manufacturers due to their ability to enhance production efficiency, reduce human labor costs, and improve product quality. They offer precision, speed, and consistency in tasks such as assembly, welding, and material handling, making them indispensable in smart factories striving for higher

productivity, quality, and competitiveness. Additionally, advancements in robotic technology, including collaborative robots (cobots) that can collaborate with human workers, have accelerated their adoption in various industries, fueling segment growth.

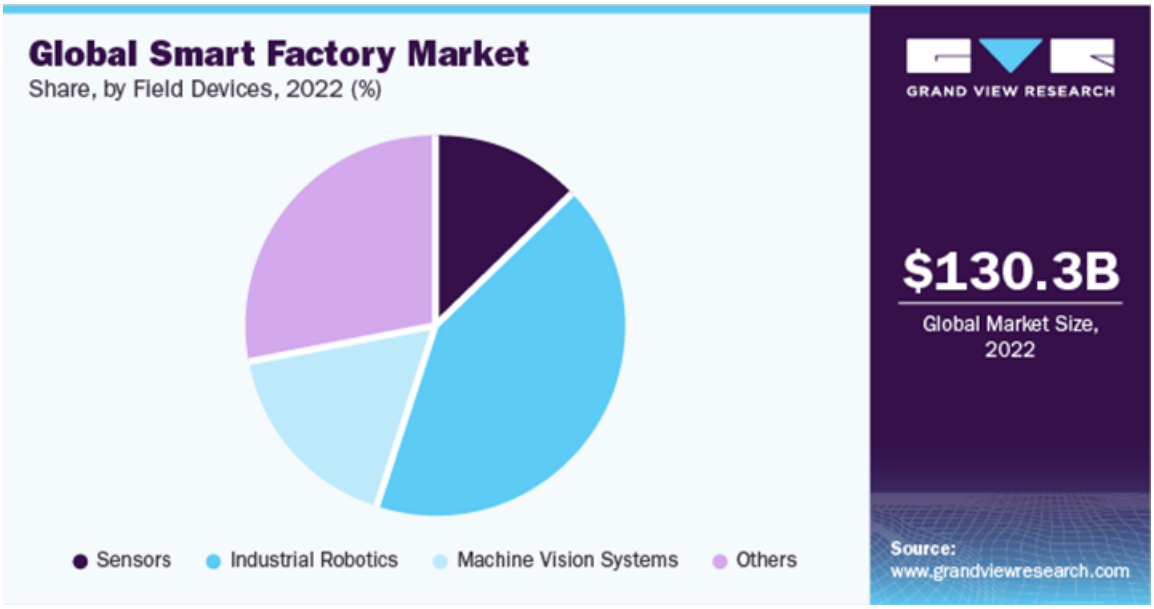

Figure 0.2: Global smart factory market 2022

Integration of IoT devices and sensor networks in smart factories research overview :

Furthermore, the integration of IoT devices and sensor networks in smart factories has been instrumental in driving these advancements. Research by MarketsandMarkets highlights that the industrial IoT market is expected to grow from USD 77.3 billion in 2021 to USD 110.6 billion by 2026, at a CAGR of 7.5%. This growth is fueled by the rising demand for real-time data analytics and predictive maintenance capabilities, which optimize asset utilization and minimize downtime.

Computer Vision Market Analysis:

In parallel, advancements in computer vision technologies have enabled significant improvements in quality control and production monitoring within smart factories. A study by Mordor Intelligence indicates that the computer vision market in manufacturing is poised to grow from USD 2.37 billion in 2020 to USD 10.7 billion by 2026, driven by applications

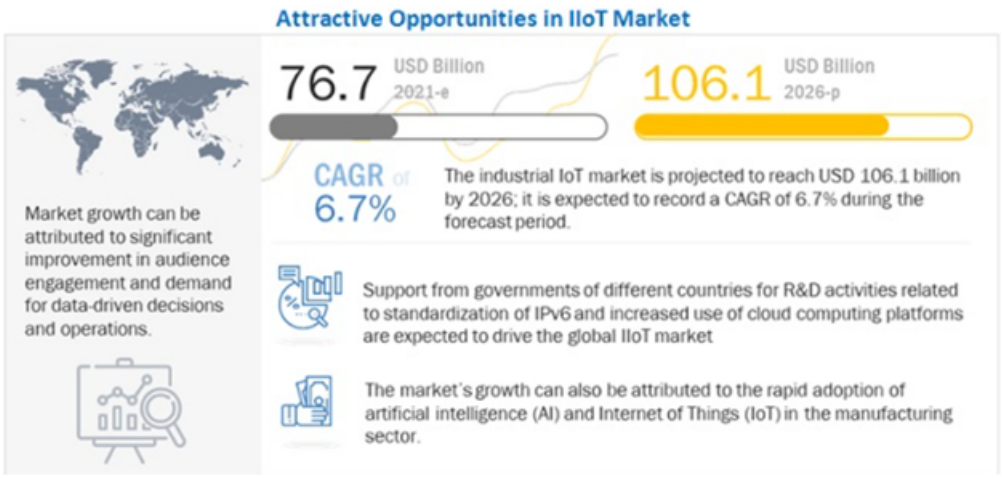

Figure 0.3: The growth of the Industrial Internet of Things (IIoT) market.

such as defect detection, product tracking, and worker safety monitoring.

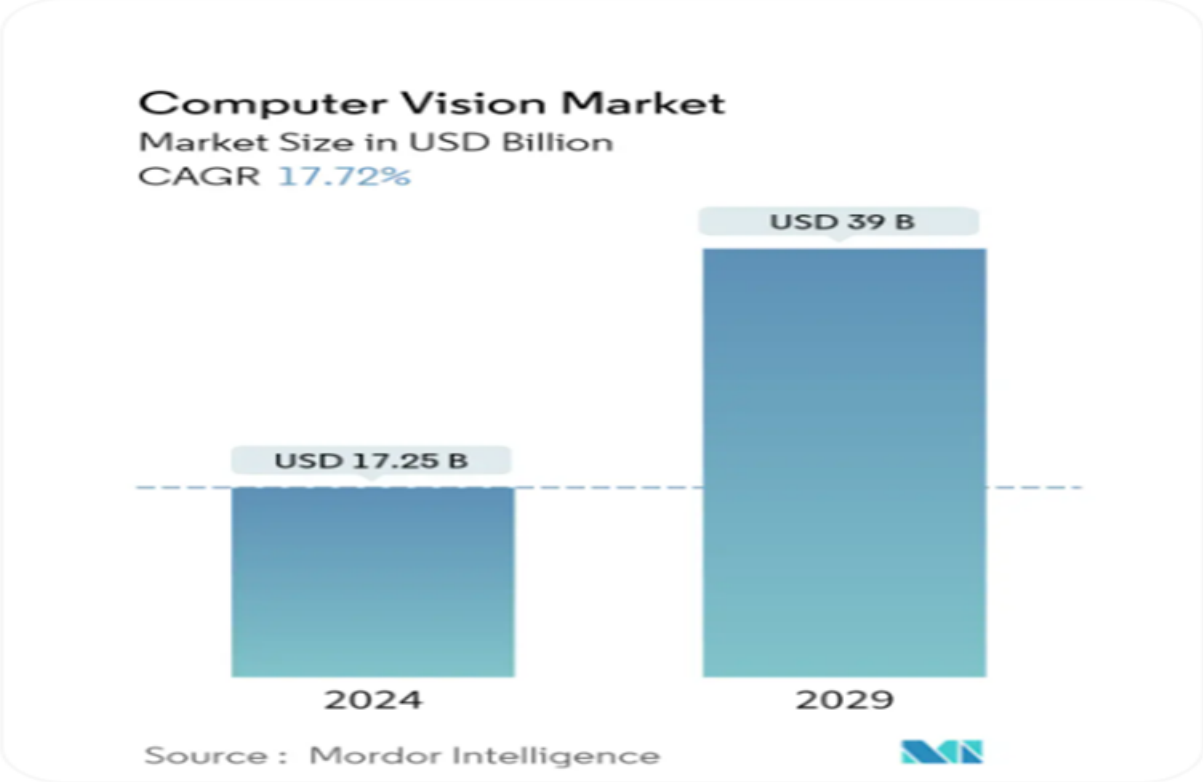

Figure 0.4: Computer vision market projected to reach 39 billion by 2029

Accidents at work report overview :

- **Number of accidents**

In 2021, there were 2.88 million non-fatal accidents that resulted in at least four calendar days of absence from work and 3 347 fatal accidents in the EU (see Table 1), a ratio of approximately 860 non-fatal accidents for every fatal accident. There was an increase between 2020 and 2021 in the total number of non-fatal accidents at work in the EU, some 150 941 more (equivalent to an increase of 5.5 %). This may, at least in part, be attributed to a return to the workplace as restrictions associated with the COVID-19 crisis were relaxed/removed. There were 11 fewer fatal accidents at work in the EU during 2021 when compared with the year before (equivalent to a decrease of 0.3 %).

- **Incidence rates**

An alternative way to analyse the information on accidents at work is to express the number of accidents in relation to the number of persons employed; this produces a ratio referred to as the incidence rate. In Figures 1 and 2, simple incidence rates are shown relating the number of accidents to the overall number of persons employed. For any given country, these statistics give an indication of the likelihood that an employed person had an accident.

These statistics underscore the momentum behind smart factory initiatives and highlight the transformative potential of integrating advanced technologies to revolutionize industrial processes. Our graduation project aims to contribute to this trajectory by implementing a cutting-edge smart factory solution that leverages these technologies to enhance operational efficiency, worker safety, and overall productivity in manufacturing environments.

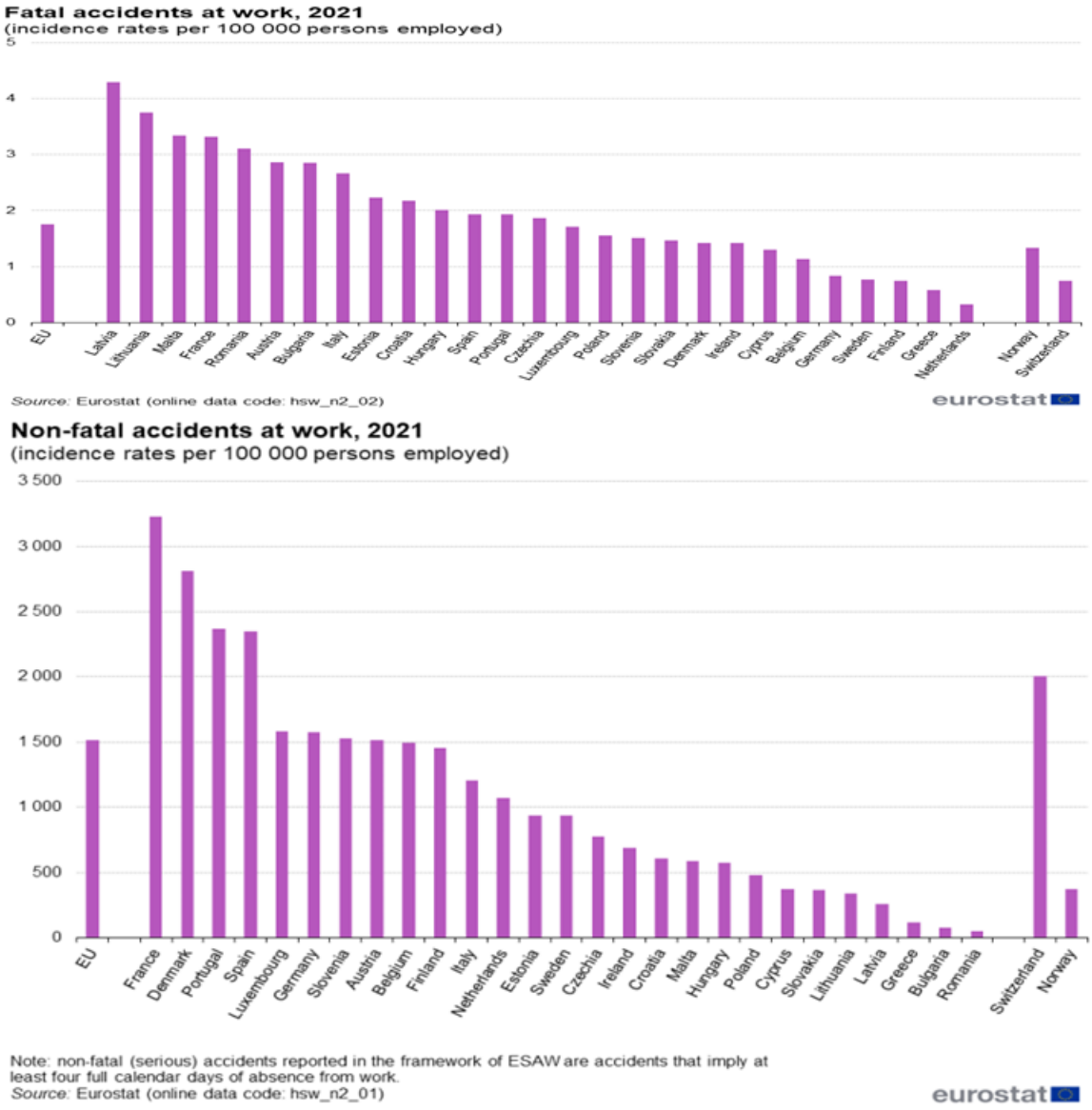

Figure 0.5: Work accidents in the EU (2021): Many more non-fatal accidents than fatal

Chapter 1

YOLO

1.1 Background

The landscape of industrial operations has undergone a profound transformation in recent years with the integration of cutting-edge computer vision technologies. Traditionally, factories relied on manual labor, periodic inspections, and static surveillance systems to monitor safety, ensure quality control, and manage assets. However, the limitations of these approaches in providing real-time insights and adapting to the dynamic nature of modern manufacturing environments have spurred a paradigm shift.

The advent of computer vision technologies has brought forth a new era in industrial automation and efficiency. By harnessing the power of advanced algorithms and image processing techniques, factories can now transcend the constraints of traditional methodologies. Among the various algorithms that have emerged, the You Only Look Once (YOLO) algorithm stands out as a beacon of innovation, offering real-time object detection capabilities that align seamlessly with the demands of the factory setting.

As industries embrace the Industry 4.0 revolution, the fusion of artificial intelligence and computer vision into manufacturing processes has become a strategic imperative. This chapter seeks to illuminate the pivotal role of computer vision, specifically spotlighting the YOLO algorithm, in reshaping the fundamental fabric of factory operations. The integration of these technologies is not merely an evolutionary step; rather, it represents a revolution in how factories conceptualize and execute tasks related to safety, quality control, and asset management.

The significance of leveraging computer vision in factories extends beyond mere automation; it encapsulates a holistic approach towards creating intelligent, adaptive, and safer work environments. Factories are

no longer confined to static surveillance systems but evolve into dynamic ecosystems where machines, equipped with vision capabilities, actively contribute to the decision-making processes.

The emphasis on safety is paramount, and computer vision serves as a proactive guardian, continuously monitoring the working conditions, identifying potential hazards, and alerting in real-time. Quality control, a cornerstone of manufacturing excellence, undergoes a metamorphosis with the precision and speed of YOLO algorithm-driven defect detection. The ability to scrutinize products in real-time not only ensures a higher standard but also enables swift corrective measures.

Asset tracking, an often underestimated facet of factory management, gains a new dimension with computer vision. YOLO's prowess in recognizing and tracking assets in real-time leads to optimized logistics, reduced downtime, and efficient resource utilization. The conventional challenges of manual tracking systems are eclipsed by the accuracy and immediacy offered by the algorithm.

In essence, this chapter serves as a gateway to the transformative journey that unfolds in subsequent sections. By delving into the historical context and the seismic shift brought about by computer vision technologies, it sets the stage for a detailed exploration of how the YOLO algorithm becomes the linchpin in fortifying safety measures, elevating product quality, and streamlining the intricate web of asset management within the dynamic tapestry of a modern factory.

1.2 Problem Statement

The traditional methodologies employed in safety monitoring and quality control within factory settings pose significant limitations that hinder the realization of optimal operational efficiency. The reliance on manual inspection processes not only consumes a considerable amount of human resources but is also susceptible to human errors and inconsistencies. Likewise, the conventional practices in asset tracking, often reliant on manual documentation or RFID systems, may fall short in providing the precision required for the seamless orchestration of logistics in a fast-paced manufacturing environment.

The labor-intensive nature of these traditional approaches introduces inefficiencies, delays, and the potential for oversight. Safety monitoring, a critical aspect of factory operations, demands constant vigilance to identify and mitigate risks promptly. The current methods, relying on periodic

checks and reactive measures, may leave workers exposed to unforeseen hazards.

Quality control, another cornerstone of manufacturing excellence, faces challenges in achieving real-time scrutiny of products. Manual inspections are time-consuming and may result in a slower production pace. Moreover, the inherent subjectivity in human evaluations can lead to inconsistencies in identifying and categorizing defects. As industries strive for higher standards and the reduction of defects, the need for a more efficient and accurate quality control mechanism becomes imperative.

In the realm of asset tracking, where the optimal utilization of resources and timely logistical maneuvers are paramount, conventional systems may lack the precision required. Manual tracking or reliance on static identifiers can lead to errors, misplacement, and inefficiencies in the supply chain. The consequence is often felt in increased downtime, delayed production cycles, and suboptimal utilization of valuable assets.

Recognizing these challenges, there is a compelling need for a comprehensive solution that not only automates these processes but does so with a level of accuracy, immediacy, and adaptability that exceeds the capabilities of traditional methods. Enter computer vision, an advanced technological paradigm that, when harnessed through sophisticated algorithms like YOLO, promises to revolutionize safety monitoring, quality control, and asset tracking in factories.

Computer vision, as a transformative force, transcends the limitations of human-centric approaches. By deploying cameras and sensors strategically throughout the factory floor, YOLO empowers the system to perceive, analyze, and respond in real-time. This paradigm shift from reactive to proactive monitoring heralds a new era where potential safety hazards are identified before they escalate, defects in products are instantaneously detected, and the location and status of assets are tracked with unprecedented precision.

In this context, the problem at hand is not just the inefficiency of existing methods but the missed opportunities for heightened safety, elevated product quality, and streamlined logistical operations. The subsequent chapters of this thesis delve into the conceptualization, development, and implementation of a solution that leverages computer vision, specifically the YOLO algorithm, to address these critical challenges and propel factory operations into a more advanced and responsive era.

1.3 Objectives

The primary objectives of this project embody a trans-formative vision for the integration of computer vision, specifically utilizing the YOLO algorithm, to revolutionize critical facets of factory operations. Each objective is intricately designed to address longstanding challenges and usher in a new era of efficiency, safety, and precision in the manufacturing environment.

1. Enhance Worker Safety through Real-time Monitoring:
   At the heart of this initiative is a commitment to transcend traditional safety monitoring methodologies. The objective strives not merely to adhere to industry safety standards but to surpass them by proactively identifying and addressing potential hazards in real-time. The deployment of computer vision transforms the factory into a dynamic environment where the YOLO algorithm acts as a vigilant guardian. Cameras strategically placed throughout the factory continuously analyze worker activities and environments. Instantaneous detection of potential risks enables swift interventions, prioritizing the well-being of workers and creating an environment where safety is not just a standard but a living, breathing aspect of daily operations.

2. Improve Product Quality Control by Swiftly Detecting and Identifying Defects:
   Quality control, a linchpin of manufacturing excellence, undergoes a revolutionary transformation with the implementation of the YOLO algorithm. The objective is not merely to meet industry standards but to set a new benchmark by leveraging the algorithm's real-time object detection capabilities. By scrutinizing products with unparalleled speed and accuracy, the system swiftly identifies and categorizes defects, ensuring that every product leaving the production line meets the highest standards of excellence. The adoption of computer vision elevates quality control from a manual and time-consuming process to a dynamic and responsive system that aligns seamlessly with the demands of modern manufacturing.

3. Optimize Asset Tracking to Ensure Efficient Logistics and Utilization
   The optimization of asset tracking represents a strategic endeavor

to enhance overall operational efficiency. The YOLO algorithm becomes the linchpin in transforming traditional asset tracking methods within the factory. Manual tracking or reliance on static identifiers is replaced by a dynamic solution that offers real-time tracking capabilities. By accurately monitoring the location and status of assets, from raw materials to finished products, the logistics process becomes more responsive and adaptive. This objective aims not only to track assets but to optimize their utilization, reducing downtime, minimizing delays, and creating a finely tuned logistical orchestra where resources are efficiently allocated, and production cycles are streamlined.

In essence, these objectives collectively form a robust framework for a comprehensive solution that leverages the advanced capabilities of computer vision, specifically the YOLO algorithm, to propel factory operations into a new era of safety, quality control, and logistical excellence. Through strategic implementation and meticulous design, this project aspires to showcase the transformative impact that advanced technologies can have on the very fabric of manufacturing processes, setting the stage for a future where efficiency and innovation coalesce to redefine the industrial landscape.

1.4 YOLO Algorithm

Abstract—In the past decade, the focus of research has shifted towards object detection due to the vast amounts of data available and the necessity to automate visual-based systems. This trend has been further fueled by advancements in computational power and Convolutional Neural Networks (CNNs) since 2012. Among the various CNN network architectures, the You Only Look Once (YOLO) network stands out for its popularity, primarily due to its speed in real-time object identification.

Introduction

In this chapter, we delve into the critical role of real-time object detection in a myriad of applications, including autonomous vehicles, robotics, video surveillance, and augmented reality. Our focus is on the YOLO (You Only Look Once) framework, which has gained prominence for its exceptional blend of speed and accuracy, enabling swift and reliable object recognition in images. We trace the evolution of the YOLO series from

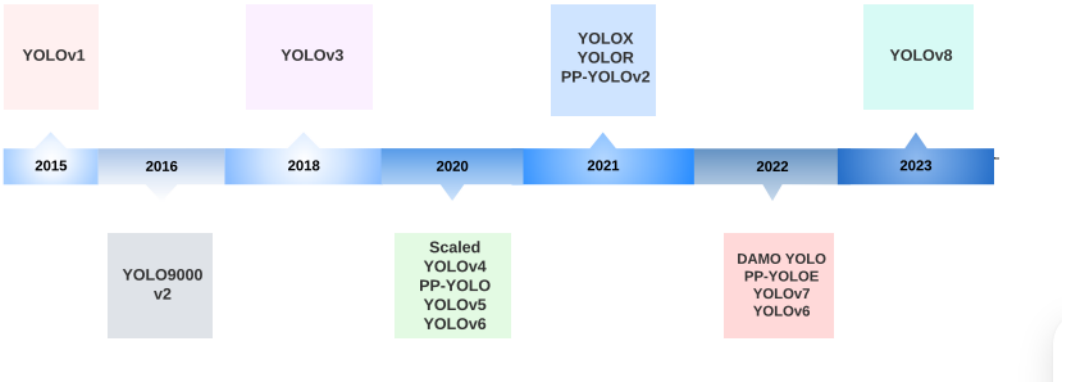

Figure 1.1: YOLO evolution

its inception, YOLOv1, to its latest iteration, YOLOv8, highlighting the key innovations, differences, and enhancements in each version. We begin with an exploration of the foundational concepts and architecture of the original YOLO model, which laid the groundwork for the subsequent advancements in the YOLO series. We then examine the improvements introduced in each version, from YOLOv2 to YOLOv8, covering aspects such as network design, loss function modifications, anchor box adaptations, and input resolution scaling. Our aim is to provide a comprehensive understanding of the YOLO framework's evolution and its impact on object detection. We also discuss the trade-offs between speed and accuracy that have characterized the framework's development, emphasizing the importance of considering the context and requirements of specific applications when choosing the most suitable YOLO model. In the context of our smart factory monitoring system project, we will provide practical examples of how we have applied these concepts and models. Lastly, we look ahead to the future of the YOLO framework, suggesting potential areas for further research and development that will continue to shape the progress of real-time object detection systems.

Object Detection Metrics and Non-Maximum Suppression (NMS)

The Average Precision (AP) is a crucial metric in assessing the effectiveness of object detection models, particularly in the context of the Common Objects in Context (COCO) dataset. Traditionally known as Mean Average Precision (mAP), this metric serves as a comprehensive evaluation tool by calculating the average precision across all object categories, yielding a

singular value for model comparison.

In the COCO dataset, there is no explicit differentiation between AP and mAP, and for the purpose of clarity, we will uniformly refer to this metric as AP throughout the remainder of this discussion.

AP is calculated by considering the precision-recall curve for each object category and then computing the area under the curve (AUC) to quantify the model's performance. This approach ensures a nuanced evaluation that takes into account both precision (the accuracy of positive predictions) and recall (the ability to detect all relevant instances).

Object detection models aim to identify and locate objects within an image, making AP a vital metric for assessing their overall efficacy. By providing a single numerical value that encapsulates the model's performance across diverse object categories, AP simplifies the comparison process between different models, facilitating informed decisions in the selection and optimization of object detection algorithms.

In conclusion, AP serves as a standardized and widely adopted metric for evaluating object detection models, contributing to the advancement of computer vision research and the development of more accurate and efficient models.

The AP metric is based on precision-recall metrics, handling multiple object categories, and defining a positive prediction using Intersection over Union (IoU).

So We to go further to understand some mentioned concepts.

- **Precision and Recall**: Precision measures the accuracy of the model's positive predictions, while recall measures the actual positive cases the model correctly identifies. There is often a trade-off between them, as detecting more objects can lead to reduced accuracy. This trade-off needs to be studied and accounted for in order to meet the model's needs, using the precision-recall curve. This curve is incorporated in the AP metric to find a balance.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **Handling multiple object categories** In the realm of object detection, where the identification and localization of multiple object

categories in an image are paramount, the Average Precision (AP) metric plays a pivotal role in evaluating the performance of detection models. The essence of this metric lies in its ability to address the nuances associated with diverse object categories.

The approach involves a meticulous calculation of the Average Precision for each individual object category present in the dataset. For every category, a precision-recall curve is constructed based on the model's predictions, capturing the trade-off between precision and recall at different confidence thresholds. The precision-recall curve is then used to compute the Average Precision (AP) – a measure that reflects how well the model performs in terms of both precision and recall for a specific category.

What makes the evaluation process even more comprehensive is the subsequent step of taking the mean of these APs across all categories. This aggregation process results in the Mean Average Precision (mAP), a metric that provides a holistic view of the model's proficiency in handling a multitude of object categories. By considering each category's performance individually and then amalgamating the results, the mAP ensures a nuanced and thorough assessment.

This meticulous categorization and evaluation process become particularly relevant when dealing with datasets that encompass a diverse range of object types. The model's ability to adapt and excel in detecting various objects is scrutinized, offering insights into its strengths and weaknesses across different categories.

In essence, the AP metric, especially when extended to mAP, goes beyond a generic performance measure. It becomes a powerful tool for model assessment, offering a fine-grained understanding of how well the model tackles the complexities posed by multiple object categories. This comprehensive evaluation is imperative in scenarios where a diverse set of objects needs to be accurately identified and localized, providing valuable insights for further model refinement and development.

$$\text{mAP} = \frac{\text{AP}_1 + \text{AP}_2 + \ldots + \text{AP}_n}{n}$$

Where:

$\text{AP}_1, \text{AP}_2, \ldots, \text{AP}_n$: Average Precision values for each individual category

n : Total number of object categories

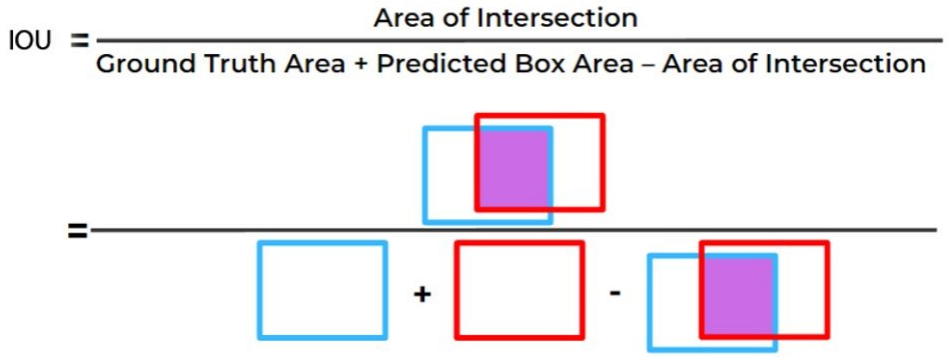

Figure 1.2: IOU Calculation, where The Ground Truth Area refers to the actual area occupied by an object in an image, as annotated manually. It serves as a reference during object detection model evaluation. In contrast, the Predicted Box Area is the area enclosed within the bounding box generated by the model.

Intersection over Union: Object detection aims to accurately localize objects in images by predicting bounding boxes. The AP metric incorporates the Intersection over Union (IoU) measure to assess the quality of the predicted bounding boxes. IoU is the ratio of the intersection area to the union area of the predicted bounding box and the ground truth bounding box (see Figure 2). It measures the overlap between the ground truth and predicted bounding boxes. The COCO benchmark[1] considers multiple IoU thresholds to evaluate the model's performance at different levels of localization accuracy.

Non-Maximum Suppression (NMS)

Object detection algorithms often produce multiple bounding boxes around the same object, each with a different confidence score. Non-Maximum

[1]COCO (Common Objects in Context): A widely utilized dataset in computer vision, COCO encompasses over 80 object categories, providing a diverse and realistic collection of images for tasks such as object detection, segmentation, keypoint detection, and captioning. Noteworthy for its complexity and scale, COCO annotations include bounding boxes, segmentation masks, keypoints, and captions, making it a comprehensive resource. The dataset has been pivotal in hosting COCO Challenges, advancing the field and introducing evaluation metrics like Average Precision (AP) with Intersection over Union (IoU) thresholds.

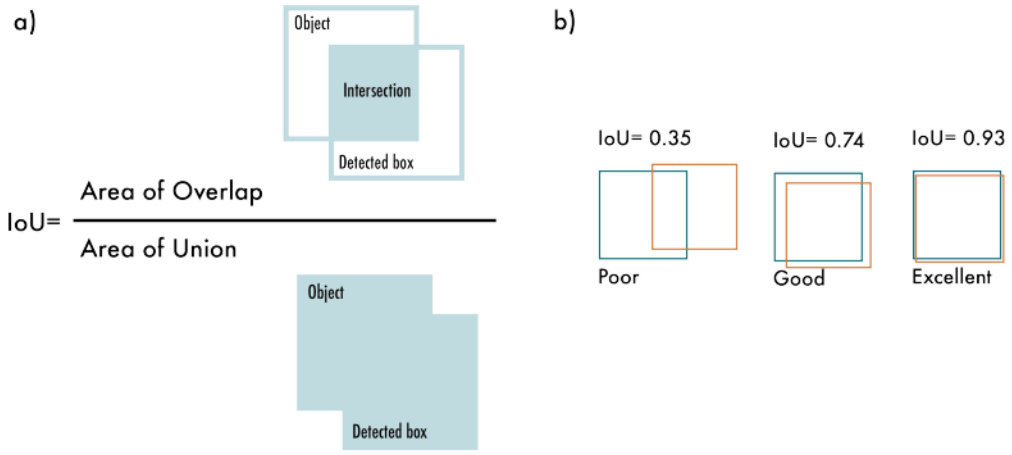

Figure 1.3: : Intersection over Union (IoU). a) The IoU is calculated by dividing the intersection of the two boxes by the union of the boxes; b) examples of three different IoU values for different box locations.

Suppression (NMS) is a post-processing technique designed to enhance the quality of object detection by addressing the issue of overlapping bounding boxes.

Key Concepts:

1. Objective of NMS:

- The primary goal of NMS is to filter out redundant and less accurate bounding boxes, retaining only the most reliable ones. This step is crucial for improving the precision and reducing redundancy in the final detection results.

2. Handling Overlapping Boxes:

- Object detection models may generate bounding boxes that overlap due to the inherent nature of the algorithm or because an object might be present at multiple scales. NMS addresses this by selecting the bounding box with the highest confidence score and suppressing others that significantly overlap with it.

3. Algorithmic Procedure:

- Algorithm 1, as mentioned in your text, outlines the typical procedure of Non-Maximum Suppression. The process involves sorting the bounding boxes based on their confidence scores. The box with the highest confidence is chosen as the initial reference, and any other box with a significant overlap is suppressed. This process is iteratively applied until all relevant boxes are considered.

4. Confidence Scores:

- Bounding boxes are associated with confidence scores reflecting the model's

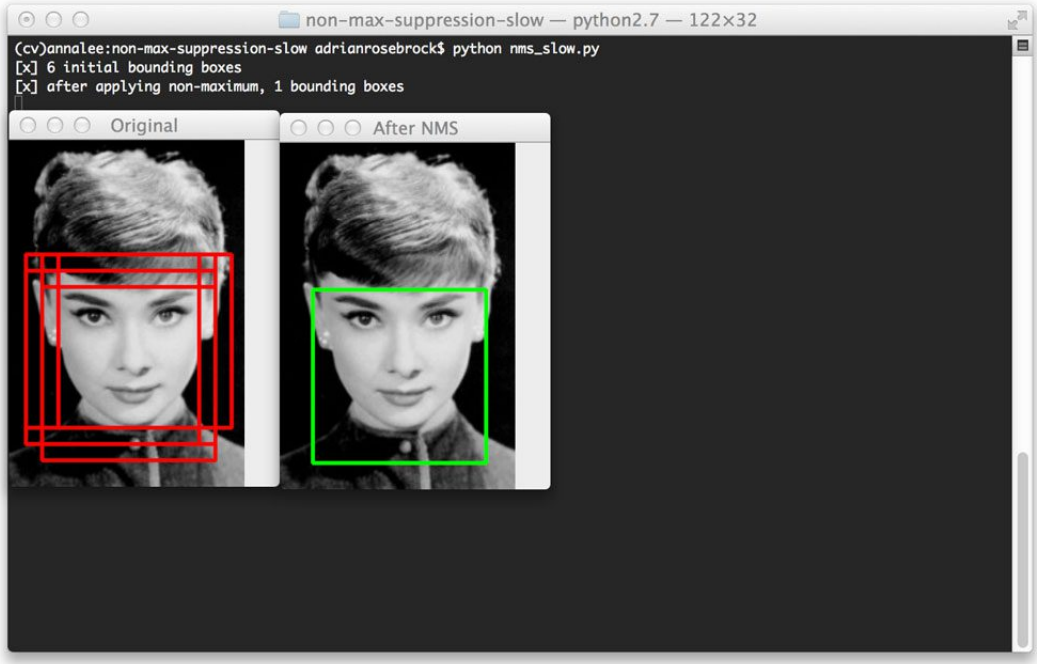

Figure 1.4: : Non-Maximum Suppression (NMS). a) Shows the typical output of an object detection model containing multiple overlapping boxes. b) Shows the output after NMS.

belief in the presence of an object within each box. NMS relies on these scores to prioritize the selection of the most reliable bounding boxes.

5. Visual Output:
- As mentioned in your text, demonstrates the typical output of an object detection model before and after applying NMS. Before NMS, there may be multiple overlapping bounding boxes with varying confidence scores. After NMS, redundant boxes are removed, and only the most accurate ones are retained.

Benefits of NMS:

- Reduced Redundancy: By eliminating redundant bounding boxes, NMS helps streamline the detection results, providing a more concise and accurate representation of detected objects.

- Improved Precision: The final output after NMS is typically more precise, as it focuses on retaining the bounding boxes with the highest confidence scores and suppressing less reliable ones.

Algorithm 1

Require:

Bounding boxes $B = \{b_1, b_2, ..., b_n\}$ where each $b_i = (x_i, y_i, w_i, h_i)$ represents a bounding box with coordinates and dimensions.

Confidence scores $C = \{c_1, c_2, ..., c_n\}$ where each c_i is the confidence score associated with the corresponding bounding box.

Ensure:

Selected bounding boxes after NMS.

procedure NMS(B, C)

    Sort the bounding boxes B based on their confidence scores C in descending order.

    Initialize an empty list, $SelectedBoxes$, to store the final selected bounding boxes.

    while B is not empty **do**

        Select the bounding box b with the highest confidence score from B.

        Add b to $SelectedBoxes$.

        Remove all bounding boxes in B that have a significant overlap (IoU $>$ threshold) with b.

    end while

    return $SelectedBoxes$ as the final output.

end procedure

How YOLO works?

YOLO is refreshingly simple. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection.

YOLO is extremely fast. Since we frame detection as a regression problem we don't need a complex pipeline. We simply run our neural network on a new image at test time to predict detections.

Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. Fast R-CNN[2]. YOLO makes less than half the number of background errors

[2]R-CNN use region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based

compared to Fast R-CNN. Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM [3] and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs.

Unified Detection:

We integrate the distinct components of object detection into a single neural network. Utilizing features from the entire image, our network predicts every bounding box and simultaneously predicts all bounding boxes across all classes. This approach allows our network to globally reason about the entire image and all its objects. YOLO's design facilitates end-to-end [4]training and achieves real-time speeds while maintaining high average precision. In our system, the input image is divided into an S $\times$ S grid, and a grid cell is assigned the responsibility of detecting an object if its center falls within that cell. For each grid cell, predictions include B bounding boxes along with confidence scores for those boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as $P_r(Object) \times IOU^{pred}_{truth}$. If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth. Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box.Each grid cell

on other objects in the scene . These complex pipelines are slow and hard to optimize because each individual component must be trained separately.

[3]Deformable Parts Models (DPM) use a sliding window approach where the classifier is run at evenly spaced locations over the entire image

[4]"End-to-end training" is a machine learning approach where a single model is trained to perform a specific task directly from raw input data to the desired output, without the need for explicit intermediate stages or handcrafted features. The entire system is optimized jointly, simplifying the training pipeline and allowing the model to autonomously learn complex representations. This approach is in contrast to traditional modular methods where tasks are divided into multiple stages handled by separate components.

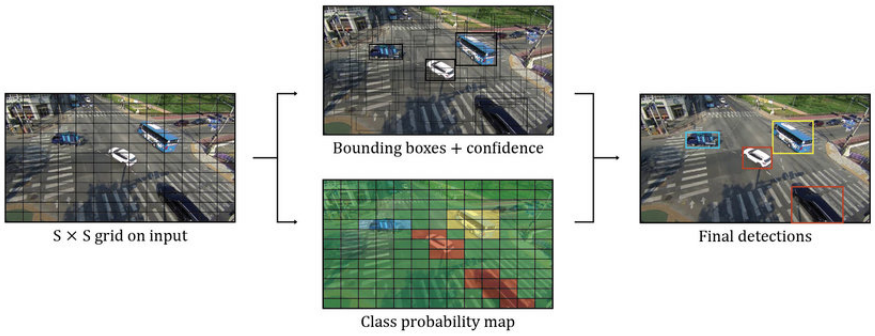

Figure 1.5: Our system models detection as a regression problem. It divides the image into an S × S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as $S \times S \times (B \times 5 + C) tensor$.

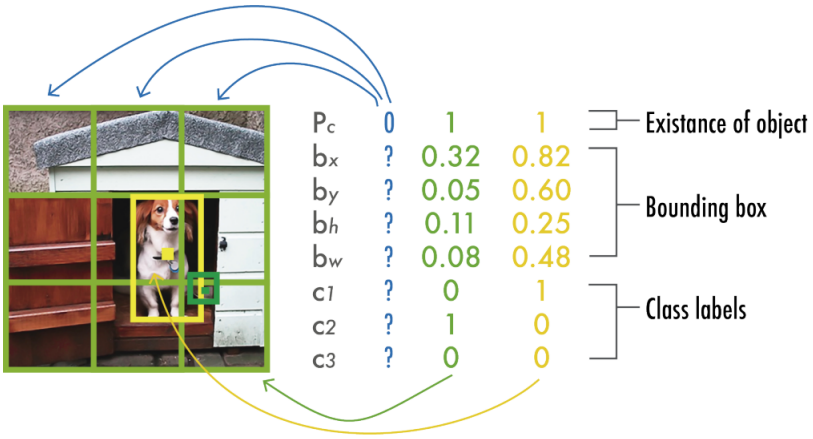

Figure 1.6: YOLO output prediction. The figure depicts a simplified YOLO model with a three-by-three grid, three classes, and a single class prediction per grid element to produce a vector of eight values

also predicts C conditional class probabilities, $Pr(Class_i|Object)$. These probabilities are conditioned on the grid cell containing an object.

$$Pr(Classi|Object) \times P_r(Object) \times IOU^{pred}_{truth} = Pr(Classi) \times IOU^{pred}_{truth}$$

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

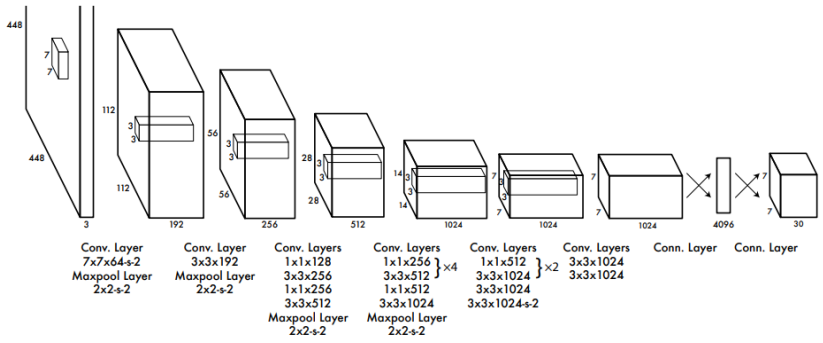

Figure 1.7: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection. The final output of our network is the $7 \times 7 \times 30$ tensor of predictions.

Network Design YOLO designed to push the boundaries of fast object detection. Fast YOLO uses a neural network with fewer convolutional layers (9 instead of 24) and fewer filters in those layers. Other than the size of the network, all training and testing parameters are the same between YOLO and Fast YOLO. See figure 1.7

YOLOv1 Training

YOLOv1 used a loss function composed of multiple sum-squared errors. In the loss function, $\lambda_{coord} = 5$ is a scale factor that gives more importance to the bounding boxes predictions, and $\lambda_{noobj} = 0.5$ is a scale factor that decreases the importance of the boxes that do not contain objects. The first two terms of the loss represent the localization loss; it computes the error in the predicted bounding boxes locations (x, y) and sizes (w, h). Note that these errors are only computed in the boxes containing objects (represented by $\mathbf{1}_{ij}^{obj}$), only penalizing if an object is present in that grid cell. The third and fourth loss terms represent the confidence loss; the third term measures the confidence error when the object is detected in the box $\mathbf{1}_{ij}^{obj}$ and the fourth term measures the confidence error when the object is not detected in the $\mathbf{1}_{ij}^{obj}$. Since most boxes are empty, this loss is weighted down by the noobj term. The final loss component is the classification loss that measures the squared error of the class conditional probabilities for each class only if the object appears in the cell $\mathbf{1}_{ij}^{obj}$

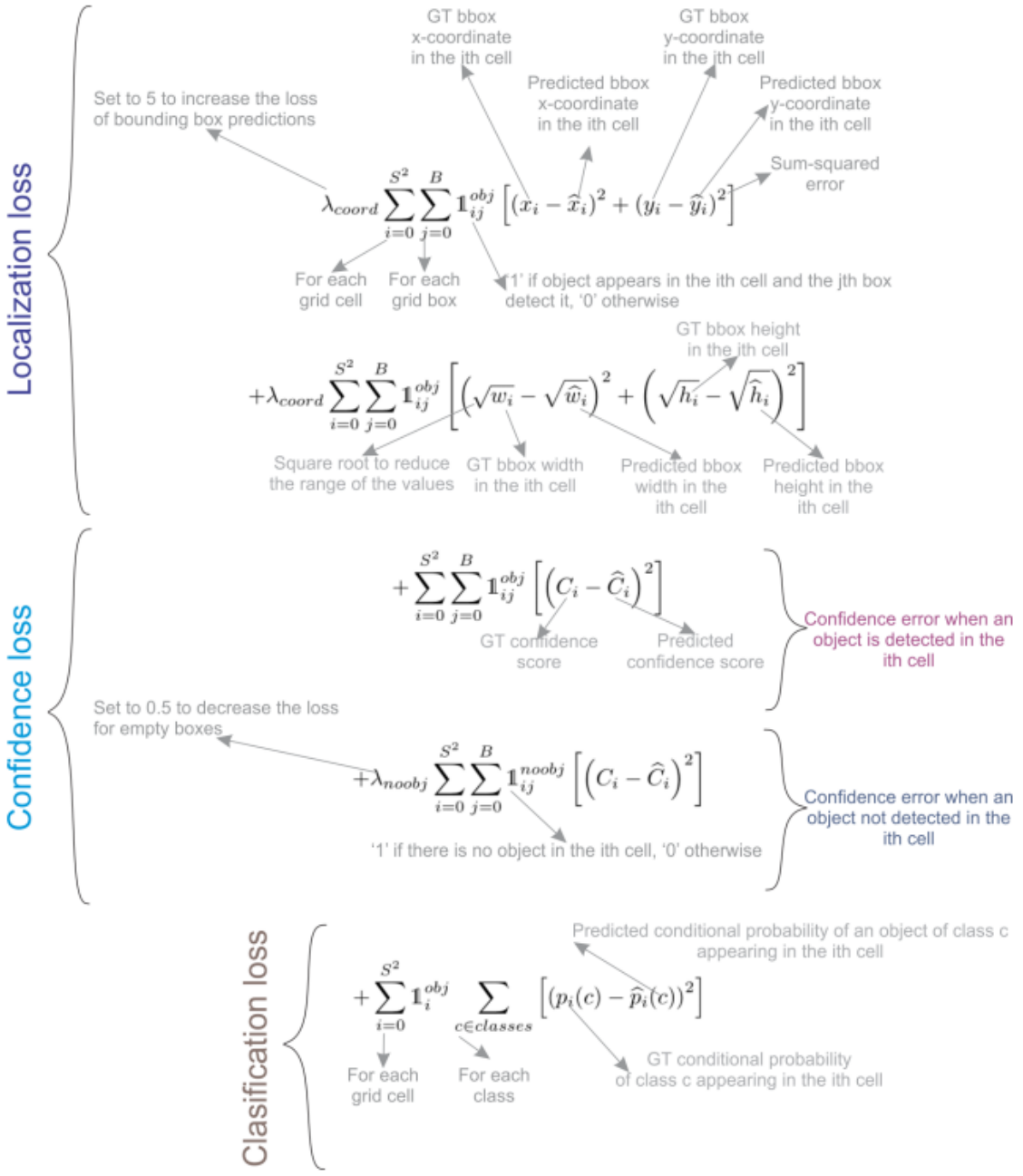

Figure 1.8: : YOLO cost function: includes localization loss for bounding box coordinates, confidence loss for object presence or absence, and classification loss for category prediction accuracy

YOLO strengths and weaknesses

The simple architecture of YOLO, along with its novel full-image one-shot regression, made it much faster than the existing object detectors allowing real-time performance. However, while YOLO performed faster than any object detector, the localization error was larger compared with state-of-the-art methods such as Fast R-CNN . There were three major causes of this limitation:
1. It could only detect at most two objects of the same class in the grid cell, limiting its ability to predict nearby.
objects.
2. It struggled to predict objects with aspect ratios not seen in the training data.
3. It learned from coarse object features due to the down-sampling layers

1.5 History of YOLO

You Only Look Once (YOLO) is a revolutionary deep learning model for object detection that has significantly impacted the field of computer vision. Developed by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, YOLO was first introduced in a paper titled "You Only Look Once: Unified, Real-Time Object Detection" in 2015. The paper presented a novel approach to object detection that significantly improved detection speed while maintaining high accuracy, making it suitable for real-time applications.

Before YOLO, object detection algorithms typically used region proposal methods to generate potential bounding boxes for objects in an image, followed by classification of these regions. This approach was computationally expensive and often resulted in slower processing speeds, especially for real-time applications.

YOLO introduced a new paradigm by framing object detection as a single regression problem, where a single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. This unified approach enabled YOLO to achieve impressive detection speeds, capable of processing images in real-time on a CPU.

Since its initial release, YOLO has undergone several iterations and improvements. YOLOv2, YOLOv3, YOLOv4, YOLOv5, YOLOv6 and more recently, YOLOv7, YOLOv8 have been developed and some other different iterations , each offering advancements in accuracy, speed, and model architecture. These iterations have made YOLO one of the most

popular and widely used object detection models, powering a wide range of applications in industries such as autonomous driving, surveillance, and healthcare.

YOLOv1

YOLOv1, or You Only Look Once version 1, was a groundbreaking deep learning model for object detection, introduced by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi in 2015. It revolutionized the field of computer vision by significantly improving the speed of object detection while maintaining competitive accuracy.

Key features of YOLOv1 include:

1. Unified Detection:
YOLOv1 framed object detection as a single regression problem, directly predicting bounding boxes and class probabilities from full images in one evaluation. This approach eliminated the need for region proposal methods used in earlier detectors, making it faster and simpler.

2. Real-Time Processing:
YOLOv1 achieved real-time processing speeds, capable of processing images at 45 frames per second (FPS) on a GPU and over 1 FPS on a CPU, making it suitable for real-time applications.

3. Strong Baseline:
Despite its speed, YOLOv1 achieved competitive accuracy, outperforming other real-time detectors at the time of its release, such as Faster R-CNN.

4. Limitations:
However,YOLOv1 had limitations, such as difficulty in detecting small objects and lower accuracy compared to state-of-the-art detectors like Faster R-CNN on some datasets, especially for fine-grained object localization.

YOLOv1 laid the foundation for subsequent versions of YOLO, which addressed many of its limitations while retaining its speed advantage. The simplicity and effectiveness of the YOLOv1 approach have made it a seminal work in the field of object detection, inspiring further research and development in real-time deep learning-based object detection.

YOLOv2

YOLOv2 was published in CVPR 2017 by Joseph Redmon and Ali Farhadi. It included several improvements over the original YOLO, to make it better, keeping the same speed and also stronger —capable of detecting 9000 categories.

The improvements were the following:

1. Darknet-19 Backbone:

YOLOv2 used the Darknet-19 architecture as its backbone, which was deeper and more powerful than the network used in YOLOv1. This allowed YOLOv2 to learn more complex features from the input images, leading to better detection performance.

2. Batch normalization:

on all convolutional layers improved convergence and acts as a regularizer to reduce overfitting.

3. High-resolution classifier:

Like YOLOv1, they pre-trained the model with ImageNet at 224×224. However, this time, they finetuned the model for ten epochs on ImageNet with a resolution of 448×448, improving the network performance on higher resolution input.

4. Fully convolutional:

They removed the dense layers and used a fully convolutional architecture.

5. Use anchor boxes to predict bounding boxes:

They use a set of prior boxes or anchor boxes, which are boxes with predefined shapes used to match prototypical shapes of objects . Multiple anchor boxes are defined for each grid cell, and the system predicts the coordinates and the class for every anchor box. The size of the network output is proportional to the number of anchor boxes per grid cell.

6. Dimension Clusters:

Picking good prior boxes helps the network learn to predict more accurate bounding boxes. The authors ran k-means clustering on the training bounding boxes to find good priors. They selected five prior boxes providing a good tradeoff between recall and model complexity.

7. Direct location prediction:

Unlike other methods that predicted offsets, YOLOv2 followed the same philosophy and predicted location coordinates relative to the grid cell. The network predicts five bounding boxes for each cell, each with five values tx, ty, tw, th, and to, where to is equivalent to P c from YOLOv1 and the final bounding box coordinates are obtained.

8. Finner-grained features:
YOLOv2, compared with YOLOv1, removed one pooling layer to obtain an output feature map or grid of 13 × 13 for input images of 416 × 416. YOLOv2 also uses a passthrough layer that takes the 26 × 26 × 512 feature map and reorganizes it by stacking adjacent features into different channelsinstead of losing them via a spatial subsampling. This generates 13 × 13 × 2048 feature maps concatenated in the channel dimension with the lower resolution 13 × 13 × 1024 maps to obtain 13 × 13 × 3072 feature maps.

9. Multi-scale training:
Since YOLOv2 does not use fully connected layers, the inputs can be different sizes. To make YOLOv2 robust to different input sizes, the authors trained the model randomly, changing the input size —from 320 × 320 up to 608 × 608— every ten batches.

With all these improvements, YOLOv2 achieved an average precision (AP) of 78.6% on the PASCAL VOC2007 dataset compared to the 63.4% obtained by YOLOv1.

Overall, YOLOv2 marked a significant advancement in the field of object detection, demonstrating that real-time, accurate object detection was achievable with deep learning models. Its innovations laid the groundwork for further improvements in subsequent versions of the YOLO series.

YOLOv3

YOLOv3 was published in ArXiv in 2018 by Joseph Redmon and Ali Farhadi. It included significant changes and a bigger architecture to be on par with the state-of-the-art while keeping real-time performance.

In the following, we described the changes with respect to YOLOv2:

1. Backbone Architecture:
YOLOv3 introduced the Darknet-53 architecture as its backbone, which is deeper and more powerful than the Darknet-19 used in YOLOv2. Darknet-53 allows YOLOv3 to extract more meaningful features from the input images, leading to improved detection performance.

2. Bounding box prediction:
Like YOLOv2, the network predicts four coordinates for each bounding box tx,ty, tw, and th; however, this time, YOLOv3 predicts an objectness score for each bounding box using logistic regression. This score is 1 for the anchor box with the highest overlap with the ground truth and 0 for

the rest anchor boxes. YOLOv3, as opposed to Faster R-CNN, assigns only one anchor box to each ground truth object. Also, if no anchor box is assigned to an object, it only incurs in classification loss but not localization loss or confidence loss.

3. Class Prediction:
Instead of using a softmax for the classification, they used binary cross-entropy to train independent logistic classifiers and pose the problem as a multilabel classification. This change allows assigning multiple labels to the same box, which may occur on some complex datasets with overlapping labels. For example, the same object can be a Person and a Man.

4. New backbone:
YOLOv3 features a larger feature extractor composed of 53 convolutional layers with residual connections.

5. Spatial pyramid pooling (SPP):
Although not mentioned in the paper, the authors also added to the backbone a modified SPP block that concatenates multiple max pooling outputs without subsampling (stride = 1), each with different kernel sizes k $\times$ k where k = 1, 5, 9, 13 allowing a larger receptive field. This version is called YOLOv3-spp and was the best-performed version improving the AP50 by 2.7%.

6. Multi-scale Predictions:
Similar to Feature Pyramid Networks, YOLOv3 predicts three boxes at three different scales.

7. Bounding box priors:
Like YOLOv2, the authors also use k-means to determine the bounding box priors of anchor boxes. The difference is that in YOLOv2, they used a total of five prior boxes per cell, and in YOLOv3, they used three prior boxes for three different scales.

YOLOv3 has been widely adopted in the computer vision community and has been used in various applications, including autonomous driving, surveillance, and robotics. Its improvements over earlier versions of YOLO have made it a popular choice for real-time object detection tasks.

YOLOv4

Two years passed, and there was no new version of YOLO. It was until April 2020 that Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao released in ArXiv the paper for YOLOv4. At first, it felt odd that different authors presented a new "official" version of YOLO; how-

ever, YOLOv4 kept the same YOLO philosophy —real-time, open source, single shot, and darknet framework— and the improvements were so satisfactory that the community rapidly embrace this version as the official YOLOv4.

We summarize the main changes of YOLOv4 in the following points:

1. Backbone Architecture:
YOLOv4 introduced the CSPDarknet53 backbone architecture, which is based on the Cross-Stage Partial network. This architecture is more efficient and powerful than Darknet-53, improving feature extraction and overall performance.

2. Improved Neck Architecture:
YOLOv4 introduced the Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PANet) modules in the neck architecture. These modules help capture multi-scale features and improve the model's ability to detect objects of different sizes.

3. Weighted-Residual Connections:
YOLOv4 introduced weighted-residual connections between layers, which helps alleviate vanishing gradient problems and improve gradient flow during training.

4. YOLOv4 Architecture:
YOLOv4 introduced the YOLOv4 architecture, which combines the backbone, neck, and head architecture into a single network. This architecture simplifies the model and improves efficiency.

5. Bag of Freebies and Bag of Specials:
YOLOv4 introduced the concepts of Bag of Freebies (BoF) and Bag of Specials (BoS), which are techniques to improve model performance. BoF includes data augmentation, label smoothing, and other techniques to improve generalization. BoS includes techniques such as Mish activation function, CIOU loss, and others to improve accuracy and convergence speed.

6. Optimization Techniques:
YOLOv4 introduced various optimization techniques, such as CutMix and DropBlock regularization, to improve model performance and robustness.

7. Hardware Acceleration:
YOLOv4 introduced support for hardware acceleration, such as NVIDIA Tensor Cores, to improve inference speed on GPUs.

YOLOv4 represents a significant improvement over YOLOv3, with en-

hanced performance, accuracy, and efficiency. It has become a popular choice for real-time object detection tasks, especially in applications where accuracy and speed are crucial.

YOLOv5

YOLOv5 was released a couple of months after YOLOv4 in 2020 by Glenn Jocher. At the time of this writing,there was no scientific paper about YOLOv5, but from the code, we know that it uses many of the improvements described in the YOLOv4 section, with the main difference being that it was developed in Pytorch instead of Darknet. YOLOv5 is open source and actively maintained by Ultralytics, with more than 250 contributors and new improvements frequently. YOLOv5 is easy to use, train and deploy. Ultralytics provide a mobile version for iOS and Android and many integrations for labeling, training, and deployment. YOLOv5 provides five scaled versions: YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), and YOLOv5x (extra large). The YOLOv5 released version at the time of this writing is v7.0, including YOLOv5 versions capable of classification and instance segmentation.

Here are the main features and improvements of YOLOv5:

1. Model Architecture:

YOLOv5 introduces a new, lightweight model architecture based on the CSPNet (Cross-Stage Partial Network) backbone and PAN (Path Aggregation Network) neck. This architecture is designed to be simple yet effective, allowing for fast and efficient object detection.

2. Model Variants:

YOLOv5 comes in four different variants: YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x, with increasing model size and performance. This allows users to choose the right balance between speed and accuracy for their application.

3. Training Pipeline:

YOLOv5 simplifies the training pipeline by providing a single command to train the model on custom datasets. It also includes features such as automatic hyperparameter tuning and mixed-precision training for improved performance.

4. Model Zoo:

YOLOv5 provides a model zoo with pre-trained models for various tasks and datasets. This makes it easy for users to start with a pre-trained model and fine-tune it on their own dataset.

5. Efficiency:
YOLOv5 is designed to be efficient, with fast inference times and low resource requirements. This makes it suitable for real-time applications and deployment on edge devices.

6. Community Support:
YOLOv5 has a large and active community of users and developers, who contribute to its development and provide support for new users.

YOLOv5 builds on the strengths of earlier versions of YOLO while introducing new features and improvements to make it a competitive choice for object detection tasks. Its simplicity, scalability, and performance have made it a popular choice among researchers and practitioners in the computer vision community.

YOLOv6

YOLO models are known for their speed and accuracy, making them highly suitable for applications requiring real-time processing. YOLOv6, introduced by Meituan in 2022, represents a significant advancement in this lineage, focusing on enhanced efficiency, speed, and deployment practicality. YOLOv6 was published in ArXiv in September 2022 by Meituan Vision AI Department. Similar to YOLOv4 and YOLOv5, it provides various models of different sizes for industrial applications.

Key Features and Improvements of YOLOv6:

1. Efficiency and Speed
YOLOv6 is optimized to deliver high-speed performance on both CPUs and GPUs. This efficiency ensures that the model can be deployed in real-time applications without significant delays. The optimization includes fine-tuning the model's architecture and training process to reduce computational overhead while maintaining high accuracy.

2. New Backbone Network - EfficientRep
A standout feature of YOLOv6 is the introduction of EfficientRep, a new backbone network designed to balance efficiency and accuracy. EfficientRep enhances feature extraction capabilities without significantly increasing the computational load, making the model both powerful and resource-efficient.

3. Advanced Training Techniques
YOLOv6 employs several advanced training techniques to improve model robustness and performance:

. Data Augmentation: Techniques such as mixup, CutMix, and Mosaic data augmentation enhance the diversity of training data, leading to a more generalized model.

. Label Smoothing: This technique is used to reduce overfitting and improve the model's generalization capabilities.

4. Improved Anchor-Based Detection

The model enhances the anchor-based detection mechanism, which improves the accuracy of bounding box predictions, particularly for objects of varying sizes. This improvement is crucial for accurately detecting and localizing objects in diverse and complex scenes.

5. Better Handling of Small Objects

YOLOv6 includes optimizations that improve the detection of small objects, addressing a common limitation in earlier versions of YOLO. This enhancement is particularly beneficial for applications where detecting small items is critical.

6. Optimized for Deployment

Designed with practical deployment in mind, YOLOv6 includes features that facilitate integration into various applications, including edge devices and mobile platforms. Its efficiency ensures that it can run effectively even in environments with limited computational resources.

Applications of YOLOv6:

1. Real-Time Object Detection

YOLOv6's high efficiency and speed make it ideal for applications requiring real-time object detection, such as video surveillance, autonomous driving, and robotics.

2. Resource-Constrained Environments

The model is optimized for deployment in environments with limited computational resources, making it suitable for devices like drones, IoT devices, and mobile phones.

3. Industrial Automation

YOLOv6 can be used in industrial settings for tasks such as quality control, inventory management, and automated inspection, where accurate and fast object detection is essential.

YOLOv6 marks a significant advancement in the YOLO family, emphasizing efficiency, speed, and practical deployment. By introducing the EfficientRep backbone network and employing advanced training techniques, YOLOv6 achieves a balance between performance and computational cost. Its improvements in small object detection and anchor-

based detection accuracy make it a powerful tool for various real-time and resource-constrained applications. YOLOv6 continues to uphold the YOLO tradition of making state-of-the-art object detection accessible and practical for a wide range of uses.

YOLOv7

he YOLO models are renowned for their balance of speed and accuracy, making them ideal for various real-time applications. YOLOv7, introduced by WongKinYiu and Alexey Bochkovskiy in 2022, represents a further evolution of the YOLO series, offering state-of-the-art performance improvements through advanced techniques and optimizations.

Key Features and Improvements of YOLOv7:

1. State-of-the-Art Accuracy :

YOLOv7 achieves state-of-the-art accuracy on several benchmark datasets, such as COCO, by incorporating novel training and architectural techniques. It surpasses the performance of its predecessors and other contemporary object detection models.

2. Efficient Backbone and Head Networks:

YOLOv7 introduces new, more efficient backbone and head networks that enhance feature extraction and detection capabilities. These networks are designed to maximize performance while minimizing computational complexity, making YOLOv7 suitable for deployment in real-time applications.

3. Re-parameterization Techniques:

YOLOv7 employs re-parameterization techniques, which involve converting complex model components into simpler, more efficient forms during inference. This approach helps to maintain high accuracy while reducing the computational load during real-time processing.

4. Improved Training Mechanisms:

YOLOv7 utilizes advanced training mechanisms, including:

.Gradient Flow Propagation: Enhances the flow of gradients during training, leading to better convergence and higher accuracy.

.New Loss Functions: Introduces improved loss functions that better capture the complexities of object detection tasks, resulting in more accurate bounding box predictions and classification.

5. Better Handling of Diverse Object Sizes:

The model includes optimizations for better handling objects of various sizes. This improvement is crucial for applications where the detection of small and large objects within the same scene is necessary.

6. Robust Data Augmentation:
YOLOv7 incorporates robust data augmentation techniques to improve model generalization and robustness. These techniques help the model perform well on a wide range of real-world scenarios.

Applications of YOLOv7:

1. Real-Time Object Detection:
YOLOv7's high efficiency and state-of-the-art accuracy make it ideal for applications requiring real-time object detection, such as autonomous driving, surveillance systems, and robotics.

2. Edge and Mobile Computing:
Due to its optimized architecture and efficiency, YOLOv7 is suitable for deployment on edge devices and mobile platforms, where computational resources are limited.

3. Industrial Automation:
YOLOv7 can be effectively used in industrial automation tasks such as quality control, inventory management, and automated inspection, where precise and fast object detection is critical.

4. Healthcare Applications:
In medical imaging and diagnostics, YOLOv7 can assist in detecting and classifying abnormalities, providing real-time analysis and support to healthcare professionals.

YOLOv7 represents a significant advancement in the YOLO series, pushing the boundaries of real-time object detection. Its state-of-the-art accuracy, efficient architecture, and advanced training techniques make it a powerful tool for a wide range of applications. YOLOv7 continues the tradition of YOLO models by balancing speed and accuracy, making it an essential choice for real-time and resource-constrained environments. With its robust performance and adaptability, YOLOv7 sets a new benchmark in the field of object detection.

YOLOv8

The YOLO series has seen continuous improvements since its inception, with each version introducing significant enhancements. YOLOv8, introduced by Ultralytics in 2023, represents the latest advancement in the series, building upon the successes of its predecessors while introducing new features and optimizations for even better performance.

Key Features and Improvements of YOLOv8:

1. Enhanced Architecture:

YOLOv8 features a redesigned architecture that improves both efficiency and accuracy. The model incorporates lessons learned from previous YOLO versions and integrates novel components to boost performance. Key architectural enhancements include:

.Improved Backbone Network: The backbone network has been optimized for better feature extraction, allowing for more accurate detection of objects in complex scenes.

.Advanced Neck Design: The neck design has been refined to improve the fusion of features from different scales, enhancing the model's ability to detect objects of varying sizes.

2. Improved Training Techniques:

YOLOv8 leverages advanced training techniques to enhance model performance and robustness:

.Data Augmentation: Utilizes sophisticated data augmentation methods to improve generalization and robustness. Techniques such as Mosaic and MixUp are employed to create diverse training samples.

.Enhanced Label Smoothing: Incorporates improved label smoothing techniques to reduce overfitting and improve generalization.

3. Efficient Inference:

YOLOv8 is optimized for efficient inference, making it suitable for real-time applications. The model achieves high throughput with low latency, enabling deployment in scenarios where quick decision-making is crucial.

4. Better Handling of Small Objects:

YOLOv8 includes specific optimizations to improve the detection of small objects. This enhancement addresses a common limitation in earlier versions, making the model more versatile for applications where small object detection is critical.

5. Robust Performance on Diverse Datasets:

YOLOv8 has been trained and tested on a variety of datasets to ensure robust performance across different types of data. This versatility makes it suitable for a wide range of applications, from industrial automation to healthcare.

6. User-Friendly Deployment:

YOLOv8 comes with improved integration capabilities, making it easier to deploy across various platforms and environments. The model is designed to be compatible with popular machine learning frameworks and can be easily integrated into cloud services, mobile devices, and edge computing

platforms.

Applications of YOLOv8:

1. Real-Time Object Detection:
YOLOv8's high efficiency and accuracy make it ideal for real-time object detection applications such as autonomous driving, video surveillance, and robotics.

2. Edge and Mobile Computing:
The model's optimized architecture and efficient inference make it suitable for deployment on edge devices and mobile platforms, where computational resources are limited.

3. Industrial Automation:
YOLOv8 can be used in industrial automation for tasks such as quality control, inventory management, and automated inspection, where precise and fast object detection is essential.

4. Healthcare Applications:
In medical imaging and diagnostics, YOLOv8 can assist in detecting and classifying abnormalities, providing real-time analysis and support to healthcare professionals.

YOLOv8 represents a significant leap forward in the YOLO family, combining enhanced architecture, advanced training techniques, and efficient inference to deliver state-of-the-art performance. Its improvements in small object detection, robust performance across diverse datasets, and user-friendly deployment capabilities make it a powerful tool for a wide range of applications. YOLOv8 continues the tradition of YOLO models by providing a balanced combination of speed and accuracy, setting a new benchmark in the field of object detection.

1.6 The future of YOLO

As the YOLO framework continues to evolve, we anticipate that the following trends and possibilities will shape future developments:

Incorporation of Latest Techniques: Researchers and developers will continue to refine the YOLO architecture by leveraging state-of-the-art methods in deep learning, data augmentation, and training techniques. This ongoing innovation process will likely improve the model's performance, robustness, and efficiency.

Benchmark Evolution: The current benchmark for evaluating object detection models, COCO 2017, may eventually be replaced by a more advanced and challenging benchmark. This mirrors the transition from the VOC 2007 benchmark used in the first two YOLO versions, reflecting the need for more demanding benchmarks as models grow more sophisticated and accurate.

Proliferation of YOLO Models and Applications: As the YOLO framework progresses, we expect to witness an increase in the number of YOLO models released each year, along with a corresponding expansion of applications. As the framework becomes more versatile and powerful, it will likely be employed in even more varied domains, from home appliances devices to autonomous cars.

Expansion into New Domains: YOLO models have the potential to extend their capabilities beyond object detection and segmentation, branching into areas such as object tracking in videos and 3D keypoint estimation. As these models evolve, they may become the foundation for new solutions that address a broader range of computer vision tasks.

Adaptability to Diverse Hardware: YOLO models will further span hardware platforms, from IoT devices to high-performance computing clusters. This adaptability will enable deploying YOLO models in various contexts, depending on the application's requirements and constraints. In addition, by tailoring the models to suit different hardware specifications, YOLO can be made accessible and effective for more users and industries.

1.7 Development and Optimization of Object Detection Models using YOLOv8

We plan to incorporate YOLOv8 into our project in the next line.

```
1 import cv2
2 import argparse
3 import pandas as pd
4 from ultralytics import YOLO
5 import cvzone
6
7 def parse_arguments():
8     """Parse command-line arguments."""
```

```
    ap = argparse.ArgumentParser()
    ap.add_argument("-v", "--video", help="path to the (optional) video file")
    args = vars(ap.parse_args())
    return args

def initialize_video_capture(video_path=None):
    """Initialize video capture object based on specified video path or camera."""
    if video_path is None:
        cap = cv2.VideoCapture(0)  # Use default camera
    else:
        cap = cv2.VideoCapture(video_path)
    return cap

def load_class_list(class_file_path):
    """Load class list from file."""
    class_list = []
    with open(class_file_path, "r") as f:
        class_list = f.read().split("\n")
    return class_list

def detect_and_track(frame, model, class_list, tracker):
    """Detect objects in a frame and track them."""
    results = model.predict(frame)
    if not results:
        return frame

    predictions = results[0].boxes.data.cpu().numpy()
    df = pd.DataFrame(predictions, columns=["x1", "y1", "x2", "y2", "confidence", "class"])

    detections = []
    for _, row in df.iterrows():
        x1, y1, x2, y2, conf, cls = row
        cls = int(cls)
        if 'person' in class_list[cls]:
            detections.append([int(x1), int(y1), int(x2 - x1), int(y2 - y1)])
```

```
44
45      bbox_id = tracker.update(detections)
46      for bbox in bbox_id:
47          x, y, w, h, id = bbox
48          cx, cy = int((x + x + w) / 2), int((y + y + h) / 2)
49          cv2.circle(frame, (cx, cy), 4, (255, 0, 255), -1)
50          cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0,
            ↪  255), 2)
51          cvzone.putTextRect(frame, f'{id}', (x, y), 1, 2)
52
53      return frame
```

In the provided Python code, several libraries are utilized to perform object detection and tracking using YOLO and OpenCV. Below is a brief summary of the role of each library:

- **OpenCV (cv2)**: OpenCV is a powerful open-source computer vision and machine learning library. It provides tools for image and video processing, including object detection, face recognition, and more. In this code, OpenCV is used to capture video frames and perform image processing operations.

- **argparse**: The argparse module makes it easy to write user-friendly command-line interfaces. It parses the command-line arguments provided by the user, allowing the code to handle input parameters for video files or other options.

- **pandas (pd)**: Pandas is a popular data manipulation and analysis library for Python. It provides data structures and functions needed to manipulate structured data seamlessly. In this code, pandas is used to handle and process the data from the object detection results.

- **YOLO (from ultralytics)**: YOLO (You Only Look Once) is a state-of-the-art, real-time object detection system. The `ultralytics` package provides an implementation of YOLO for detecting objects within images and video frames. This library is used for running the object detection model on the input frames.

- **cvzone**: CVZone is a library that simplifies computer vision tasks, particularly those involving OpenCV. It provides easy-to-use functions for common tasks such as displaying text, shapes, and bounding

boxes on images. In this code, cvzone is used to draw bounding boxes and other annotations on the video frames for detected objects.

Training a Computer Vision Model Using YOLO

In the domain of computer vision, numerous methodologies exist for training models. One prominent and effective approach involves using the YOLO (You Only Look Once) object detection framework. Below is a detailed, formal explanation steps of this training process, focusing on key aspects such as model initialization, training, and evaluation.

1- Installing tools "ANACONDA"

Anaconda is a popular open-source distribution of the Python and R programming languages for scientific computing, specifically for data science, machine learning, and large-scale data processing. Here are its key features and components:

Package Manager (Conda): • Conda: A versatile package manager that simplifies installing, updating, and managing software packages and their dependencies. It can handle both Python and non-Python packages.
• Environments: Conda allows the creation of isolated environments to manage different versions of libraries and dependencies, which is crucial for maintaining project consistency.

Pre-installed Libraries:

Anaconda comes with over 1,500 pre-installed data science and machine learning libraries, including popular ones like NumPy, pandas, scikit-learn, TensorFlow, PyTorch, and Matplotlib. This makes it easier to get started without worrying about complex installations and dependency issues.

Anaconda Navigator:

A graphical user interface (GUI) that simplifies the management of packages, environments, and applications. Users can launch applications, manage environments, and install or update packages without needing to use the command line.

Common Uses of Anaconda

1. Data Science and Machine Learning: Anaconda provides all the necessary tools and libraries for data analysis, visualization, and building machine learning models. It's widely used in both academia and industry.
2. Environment Management: Developers often need to work on multiple projects with different dependencies. Anaconda's environment management capabilities make it easy to create and switch between isolated environments, ensuring that projects do not interfere with each other.
3. Reproducibility: By encapsulating dependencies within isolated environments, Anaconda ensures that projects can be easily shared and reproduced on different systems.
4. Education and Training: Anaconda is popular in educational settings for teaching data science and programming due to its comprehensive set of tools and ease of use.

2- Setup The Environment

"conda create -p path python=version" is a command used with the Conda package manager to create a new Conda environment at a specified path with a specified version of Python and set up the necessary files and directories for a Conda environment.

Advantages of Using the command where "-p" flag stands for "prefix":

• Custom Location: You have the flexibility to create environments in any directory, which can be useful for organizing environments by project or for storage management.
• Isolation: Environments created in custom locations are isolated from the default Conda environments, reducing the risk of conflicts.
• python=version: This specifies the Python version you want to install in the new environment. For example, python=3.8 would install Python 3.8 in the new environment.
• After activate the Conda environment, we can install any library inside will not conflict with other library in PC. Then write the command "conda activate path of environment created" to activate the environment.

3- A way to collect dataset

Write command "simple-image-download= = 0.4" in Anaconda prompt. Then writing following python code for getting 100 images represent con-

struction workers for example: -

```
from simple_image_download import simple_image_download as
 ↪  sid
response=sid.simple_image_download
keywords = ['construction workers']
for kw in keywords:
    response().download(kw,100)
```

The accuracy of a model tends to improve as the size of the dataset increases. This phenomenon can be attributed to several factors inherent to larger datasets.

I. Increased Representation: A larger dataset typically covers a more comprehensive range of instances, ensuring better representation of the underlying population. With a diverse set of examples, the model becomes exposed to a wider array of patterns, leading to a more robust and accurate understanding of the data distribution.
II. Reduced Bias: A larger dataset helps mitigate the influence of bias that might be present in smaller samples. Biases can arise due to various factors, such as imbalanced class distributions or disproportionate representation of certain features. By collecting a larger dataset, the chances of capturing a more balanced representation of the target population increase, resulting in reduced bias and enhanced accuracy.
III. Generalization Capability: A model trained on a larger dataset gains the ability to generalize better to unseen data. The increased variety of instances in a larger dataset allows the model to learn more diverse patterns, enabling it to make accurate predictions or classifications on new, unseen examples. This improved generalization capability is crucial for achieving higher accuracy in real-world scenarios.
IV. Reduced Overfitting: Overfitting occurs when a model becomes too specialized in learning the peculiarities of the training data, leading to poor performance on new data. Larger datasets provide a wider range of training examples, discouraging the model from memorizing specific instances and instead encouraging it to learn more generalizable patterns. Consequently, the risk of overfitting decreases, resulting in improved accuracy on unseen data.
V. Statistical Significance: With a larger dataset, statistical significance is more likely to be achieved. It allows for more reliable estimations of model

performance metrics, such as accuracy, precision, recall, or F1 score. This statistical robustness enhances confidence in the reported accuracy, making it a more trustworthy indicator of the model's true performance.

4- Annotating or Labeling images

The command: "pip install labelImg" is used to install the LabelImg software package via the Python package manager, pip.
pip: This is the package installer for Python. It allows you to install and manage additional libraries and dependencies that are not included in the standard Python distribution.

pip stands for "Pip Installs Packages" and is the de facto standard for installing Python packages. It is a command-line tool that allows you to install, uninstall, upgrade, and manage packages from the Python Package Index (PyPI) and other package repositories.

What is LabelImg?
LabelImg is a graphical image annotation tool commonly used for labeling images for object detection and image classification tasks in machine learning. It allows users to draw bounding boxes around objects in images and assign labels to those objects. The annotations are typically saved in XML files in PASCAL VOC format or as TXT files in YOLO format, which are commonly used formats for training machine learning models.

Features of LabelImg:

1. User-Friendly Interface: Provides an easy-to-use graphical interface for labeling images.
2. Supports Multiple Formats: Can save annotations in PASCAL VOC format (XML) or YOLO format (TXT).
3. Customizable: Users can customize the tool to fit their specific needs, including changing the label names and adding keyboard shortcuts for efficiency.
4. Open Source: LabelImg is open-source software, allowing for community contributions and modifications.

Then open labelImg:

And create labels you want by drawing bounding boxes around objects

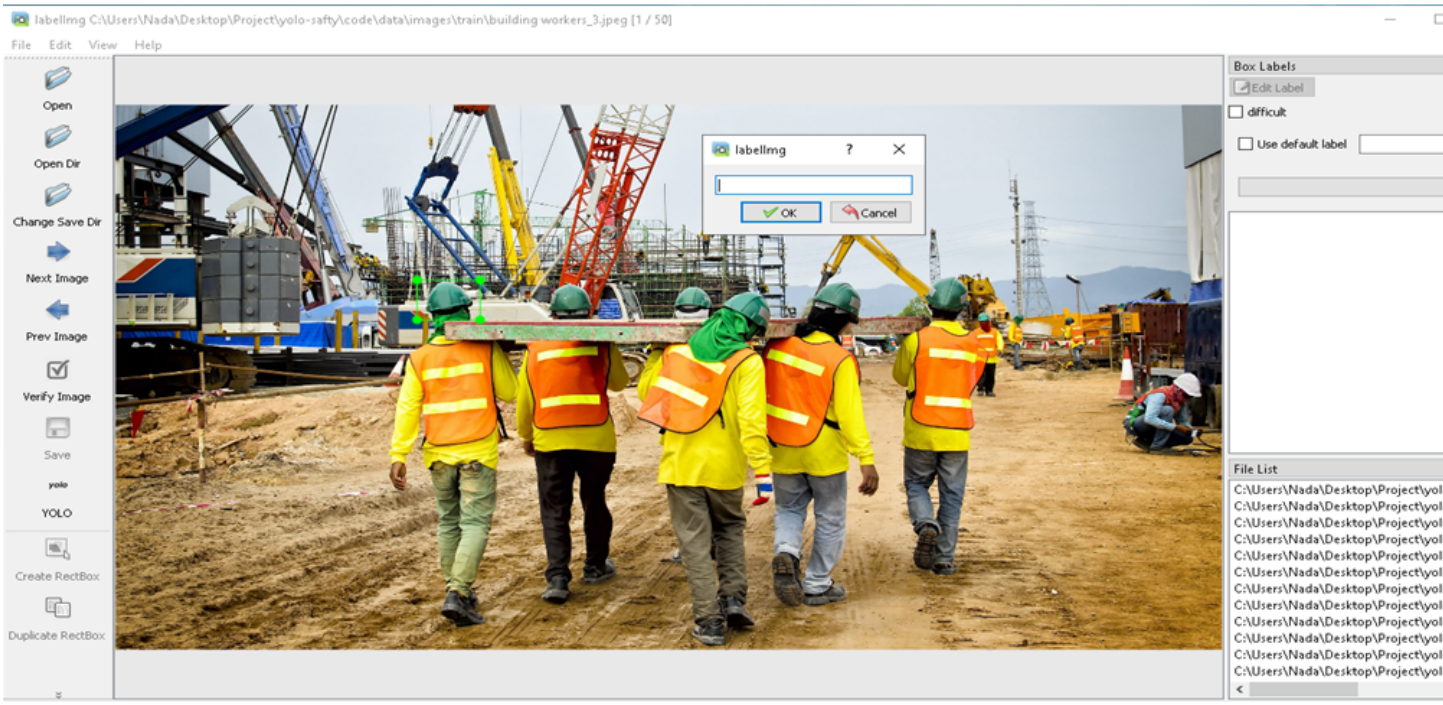

Figure 1.9: open labelImg

in images and assigning labels to those objects.

5- Setup YOLOv8

Write the command “pip install ultralyticsplus” in Anaconda prompt to install ultralyticsplus in environment.

Ultralytics is the organization behind the YOLO (You Only Look Once) series of object detection models. YOLOv8 is the latest iteration in the YOLO series, building upon the advancements made in previous versions such as YOLOv5. UltralyticsPlus might refer to a suite of tools or enhancements provided by Ultralytics, possibly including advanced features, support, or proprietary enhancements to their open-source offerings.

Potential Functionalities of ”ultralyticsplus”

1. Enhanced YOLO Models: Could include advanced versions of YOLO models, such as YOLOv8, with additional features or optimizations.
2. Advanced Tools and Utilities: Tools for model training, evaluation, and deployment that go beyond the standard YOLO implementations.

3. Integration Features: Enhanced capabilities for integrating YOLO models with various applications, platforms, and frameworks.
4. Premium Features: Access to proprietary features or models that are not available in the standard open-source offerings.
5. Support and Documentation: Additional resources, documentation, and possibly access to premium support from Ultralytics.

Write the command: pip3 install - - upgrade torch torchvision torchaudio –index-url https://download.pytorch.org/whl/cu121

Purpose of the Command:

1. Installation of PyTorch and Related Packages: This command installs or upgrades the PyTorch library (torch), along with its associated libraries torchvision (for computer vision) and torchaudio (for audio processing).
2. GPU Support with CUDA 12.1: By specifying the –index-url, it ensures that the versions of the packages being installed are compatible with CUDA 12.1, allowing for GPU acceleration if your system has a compatible NVIDIA GPU and CUDA drivers installed.

Why Use This Command?

1. Instead of utilizing the CPU version, which tends to exhibit sluggish performance, it is recommended to employ the GPU version. This alternative computing approach harnesses the power of the Graphics Processing Unit (GPU), which is specifically designed to handle parallel processing tasks more efficiently. By utilizing the GPU version, computational tasks can be executed significantly faster, resulting in improved overall performance and productivity.
2. GPU Acceleration: Installing the CUDA-enabled versions of these packages allows you to leverage NVIDIA GPUs for faster computations, which is crucial for training large neural networks and processing large datasets efficiently.
3. Up-to-date Software: The –upgrade flag ensures that you are installing the latest versions of these packages, which include the newest features, improvements, and bug fixes.
4. Compatibility: Specifying the –index-url ensures that you are downloading package versions that are specifically built to be compatible with CUDA 12.1, preventing compatibility issues that might arise from mis-

matched versions.
- Create data-config.yaml file in Visual Studio Code “VS code”

This file is likely used to store configuration settings or parameters related to data processing or data-related operations within an application or system. It provides a structured and easily readable format for specifying various data-related configurations, such as file paths, data sources, data formats, data transformations, or any other relevant settings. By using a YAML file like ”data-config.yaml,” developers or users can easily modify or customize the data-related configurations without having to modify the actual application code. This allows for greater flexibility and easier management of data-related settings, making it simpler to adapt the application to different environments or use cases.

For example, yaml file for Safety Workers:

```yaml
path: 'C:\Users\Nada\Desktop\Project\yolo-safty\code\data_2'
train: train/images
val: valid/images
test: test/images

nc: 5
names: ['fall', 'Safty-Vest', 'Helmet', 'without_Helmet',
 ↪  'without_Safty-Vest']
```

YAML file provides the necessary configuration information for the project’s dataset, including the file path, paths to training, validation, and testing images, the number of classes, and the corresponding class labels. This information is crucial for training and evaluating object detection models using the YOLO algorithm.

6- Train the model

- for example write ”yolo task=detect mode=train epochs=50 data=data-config.yaml model=yolov8n.pt imgsz=640 workers=1 batch=3” in terminal of VS code

Purpose of Each Parameter:
- task=detect: Defines the purpose of the model training, which is to

detect objects within images.
• mode=train: Indicates that the command should run in training mode, where the model will learn from the data.
• epochs=50: Determines how many times the training algorithm will iterate over the entire dataset, influencing how well the model learns.
• data=data-config.yaml: Provides the necessary configuration for the dataset, ensuring the model knows where to find the training and validation data and how to interpret it.
• model=yolov8n.pt: Selects the specific YOLOv8 nano model architecture and optionally loads a pre-trained model to start training from a pre-learned state.
• imgsz=640: Sets the input image size, affecting the trade-off between model accuracy and computational efficiency.
• workers=1: Specifies the number of worker threads for loading data, impacting the data loading speed and overall training efficiency.
• batch=3: Defines the number of images processed in each batch, impacting memory usage and training dynamics.

Model	Size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed GPU (ms)	Params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 1.10: Types of yolov8 Models:we use yolov8n to make train Small, fast, less accurate, suitable for edge devices.

7- Evaluate and Test

When training a model, observing a decreasing loss and increasing precision (or maintaining a high precision) are positive signs. They indicate that the model is learning effectively from the data, improving its predictive accuracy, and becoming more reliable for making decisions or predictions on new, unseen data. These metrics are critical for assessing and validating the quality of a trained machine learning model.

Parameter	Purpose	Effect of Increasing	Effect of Decreasing
epochs	Number of times the model sees the entire dataset	Better learning, longer training, risk of overfitting	Faster training, less learning, risk of underfitting
workers	Number of CPU threads for data loading	Faster data loading and training, better CPU utilization	Slower data loading, potential training bottleneck
batch	Number of samples per training iteration	More stable and faster training, better hardware utilization	Less memory usage, potentially noisier updates, slower training

Figure 1.11: The purpose of some parameters.

- test images by the training model result best.pt for example:

```
from ultralytics import YOLO

model = YOLO('./runs/detect/train8/weights/best.pt')

model.predict('./data/images/train/building workers_6.jpeg',
 ↪  save=True, show=True, conf=0.7 , save_txt=True)
```

When training an object detection model, such as YOLO, you often encounter different weight files saved during the training process. The files best.pt and last.pt represent different stages of the model's training. Here's the difference between them and why you might choose to use best.pt over last.pt:

"best.pt"
- Definition: The best.pt file contains the weights of the model that achieved the best performance on the validation set during training.
- Selection Criteria: The performance is typically measured using metrics such as mAP (mean Average Precision), loss, or a combination of other evaluation metrics. The model state corresponding to the highest evaluation score or lowest validation loss is saved as best.pt.
- Purpose: Using best.pt ensures that you are using the model weights that were most effective in terms of validation performance. This is crucial for generalizing well to unseen data and is usually the preferred model for inference.

“last.pt”
- Definition: The last.pt file contains the weights of the model at the final epoch of training.
- Selection Criteria: It simply represents the state of the model at the very end of the training process, regardless of its performance on the validation set.
- Purpose: last.pt can be useful if you want to continue training the model from where it left off or if you want to analyze the final state of the model. However, it might not be the best performing model in terms of validation metrics.

Why Use best.pt

• Generalization: Since best.pt is chosen based on the best validation performance, it is more likely to generalize better to new, unseen data. This means it will probably perform better in real-world applications.
• Avoiding Overfitting: Training might lead to overfitting towards the end, where the model becomes too tailored to the training data, resulting in decreased performance on validation or test data. best.pt helps in mitigating this issue as it captures the optimal performance point during training.

Explanation of “model.predict”:

• Image Prediction:
model.predict(’./data/images/train/building workers-62.jpeg’: This line performs object detection on the image specified by the path ./data/images/-train/building workers-6.jpeg. The YOLO model will detect objects within this image.

• Arguments:
- save=True: Saves the annotated image with bounding boxes drawn around detected objects.
- show=True: Displays the annotated image with bounding boxes.
- conf=0.7: Sets the confidence threshold for object detection to 0.7. This means that only detections with a confidence score of 0.7 or higher will be shown.
- save-txt=True: Saves the detections in a text file. This typically includes the class of the detected object, confidence score, and bounding box coor-

dinates.

Worker Safety Monitoring

Workplace safety is the prevention of hazards, accidents, and other negative effects in the workplace. It encompasses a set of regulations, practices, and measures designed to reduce dangers, accidents, and injuries in the workplace. Implementing a robust occupational health and safety policy is crucial for both employee well-being and the financial health of a business.

Wear Proper Safety Equipment

In this section, the focus is on using YOLOv8 to monitor worker safety in factory environments, specifically detecting the usage of coveralls and vests as essential safety equipment.

1. **YOLOv8 Configuration for Safety Attire Detection**:
    - **Model Architecture**: YOLOv8 is configured with an optimized backbone and detection heads tailored for accurately detecting safety attire such as coveralls and vests in factory environments. The architecture is designed for both speed and accuracy, crucial for real-time monitoring.
    - **Training Setup**: Annotated datasets of industrial scenes featuring workers wearing coveralls and vests were utilized for training the YOLOv8 model. These datasets are essential for training the model to recognize and distinguish safety equipment effectively.
    - **Training Parameters**: Various training parameters were optimized for effective model training:
        - Epochs: The model was trained over multiple epochs to improve its ability to detect safety attire under diverse conditions.
        - Learning Rate: Optimized learning rate schedules were applied to ensure efficient convergence during training.
        - Batch Size: Carefully chosen batch sizes were used to balance training speed and memory usage.

- Augmentation Techniques: Techniques such as random crop and horizontal flip were employed to augment the training data, enhancing the model's robustness against different orientations and scenarios encountered in factory settings.

2. **Code Snippet**:

```
    def ObjectPredictor():
    """Process video frames for object detection."""
    args = parse_arguments()
    video_path = args.get("video", None)

    helmet_vest_model_path =
    ↪  r'C:\..\VestHelmet_Detection\best.pt'
    drowsy_model_path =
    ↪  r'C:\..\Awakeness_Detection\best.pt'

    helmet_vest_classnames =['fall', 'Safty-Vest',
    ↪  'Helmet', 'without_Helmet',
    ↪  'without_Safty-Vest']
    drowsy_classnames = ['drowsy', 'awake', 'fainted']

    helmet_vest_model = YOLO(helmet_vest_model_path)
    drowsy_model = YOLO(drowsy_model_path)
    cap = initialize_video_capture(video_path)

    if not cap.isOpened():
        print("Error opening video capture.")
        return

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Process frame with both models
        frame = Safety_frame(frame, helmet_vest_model,
        ↪  helmet_vest_classnames)
        frame = Safety_frame(frame, drowsy_model,
        ↪  drowsy_classnames)
```

```
28
29          cv2.imshow("Object Detection", frame)
30
31          if cv2.waitKey(1) & 0xFF == 27:  # Exit on
            ↪  'Esc' key
32              break
33
34      cap.release()
35      cv2.destroyAllWindows()
36
37  if __name__ == "__main__":
38      ObjectPredictor()
```

3. **Results Visualization**:

   Visual representations of the YOLOv8 model's performance in detecting coveralls and vests among workers in factory settings. These visualizations demonstrate the model's effectiveness in enhancing workplace safety through automated monitoring of safety equipment compliance.

Below is a sample image showcasing the detection results of the YOLOv8 model for safety attire in industrial settings. Figure 1.12 illustrates the detection of safety gear using YOLOv8, where the model accurately identifies workers wearing helmets and vests, crucial for ensuring workplace safety in industrial environments.

This structured approach outlines how YOLOv8 was implemented for detecting safety attire, including relevant code snippets and visual results demonstrating the model's performance in real-world scenarios.

State Detection of Individuals: Awake, Asleep, or Unconscious

Knowing the state of workers plays a pivotal role in fostering a safe, productive, and compliant workplace environment. By leveraging advanced

Figure 1.12: Detection of safety gear using YOLOv8: The model accurately identifies workers wearing helmets and vests, crucial for ensuring workplace safety in industrial environments.

technologies and proactive monitoring strategies, organizations can effectively manage risks, enhance worker well-being, and optimize overall operational performance.

1- Model Setup for State Detection

- **Model Architecture**: Utilized a deep learning model trained on labeled datasets to classify individuals into awake, asleep, or unconscious states. The architecture incorporates advanced features to accurately distinguish between these states in real-world scenarios.

- **Training Data**: Annotated datasets included images or video frames depicting individuals in various states (awake, asleep, unconscious). These datasets are essential for training the model to recognize and classify different physiological states.

- **Training Process**: Employed transfer learning using a pre-trained model (e.g., ResNet, VGG) to benefit from existing knowledge and improve model accuracy. Fine-tuning was performed on the specific task of state detection to adapt the model to the nuances of workplace environments.

2- Code Snippet

```
    def ObjectPredictor():
    """Process video frames for object detection."""
    args = parse_arguments()
    video_path = args.get("video", None)

    helmet_vest_model_path =
    ↪  r'C:\..\VestHelmet_Detection\best.pt'
    drowsy_model_path = r'C:\..\Awakeness_Detection\best.pt'

    helmet_vest_classnames =['fall', 'Safty-Vest', 'Helmet',
    ↪  'without_Helmet', 'without_Safty-Vest']
    drowsy_classnames = ['drowsy', 'awake', 'fainted']

    helmet_vest_model = YOLO(helmet_vest_model_path)
    drowsy_model = YOLO(drowsy_model_path)
    cap = initialize_video_capture(video_path)

    if not cap.isOpened():
        print("Error opening video capture.")
        return

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Process frame with both models
        frame = Safety_frame(frame, helmet_vest_model,
        ↪  helmet_vest_classnames)
        frame = Safety_frame(frame, drowsy_model,
        ↪  drowsy_classnames)

        cv2.imshow("Object Detection", frame)

        if cv2.waitKey(1) & 0xFF == 27:  # Exit on 'Esc'
        ↪  key
            break
```

```
33
34      cap.release()
35      cv2.destroyAllWindows()
36
37  if __name__ == "__main__":
38      ObjectPredictor()
```

3- Results Visualization

Sample Output Images: Below Figure 1.13 is a sample image demonstrating the detection results of the model for awake and unconscious states.

Figure 1.13: Model Detection of Awake and Fainted Persons: Highlighting the capability to distinguish between alert individuals and those requiring immediate attention, ensuring timely intervention for enhanced workplace safety.

This structure outlines how a deep learning model can be implemented and utilized for detecting different states of individuals (awake, asleep, or unconscious), including relevant code snippets and visual results showcasing the model's performance. Adjust the code and preprocessing steps as per your specific model and data requirements.

Crowd Detection

This section focuses on the deployment of YOLOv8 for crowd detection within factory settings, highlighting its role in enhancing workplace safety protocols and operational efficiency.

1- YOLOv8 Configuration for Crowd Detection:

• Model Architecture: Configuration of YOLOv8 tailored for accurately detecting and monitoring crowds in industrial environments.
• Training Dataset: Utilization of annotated datasets containing images or video frames depicting crowded factory floors for robust model training.
• Training Process: Fine-tuning of model parameters and integration of tracking algorithms to ensure precise crowd detection and management.

2- Code snippet:

```
def detect_and_track(frame, model, class_list, tracker):
    """Detect objects in a frame and track them."""
    results = model.predict(frame)
    if not results:
        return frame

    predictions = results[0].boxes.data.cpu().numpy()
    df = pd.DataFrame(predictions, columns=["x1", "y1",
    ↪  "x2", "y2", "confidence", "class"])

    detections = []
    for _, row in df.iterrows():
        x1, y1, x2, y2, conf, cls = row
        cls = int(cls)
        if 'person' in class_list[cls]:
            detections.append([int(x1), int(y1), int(x2 -
            ↪  x1), int(y2 - y1)])

    bbox_id = tracker.update(detections)
    for bbox in bbox_id:
        x, y, w, h, id = bbox
        cx, cy = int((x + x + w) / 2), int((y + y + h) / 2)
        cv2.circle(frame, (cx, cy), 4, (255, 0, 255), -1)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0,
        ↪  255), 2)
```

```
23          cvzone.putTextRect(frame, f'{id}', (x, y), 1, 2)
24
25      return frame
26
27  def CrowdDetector():
28      """Process video frames for crowd detection."""
29      args = parse_arguments()
30      video_path = args.get("video", None)
31
32      class_file_path =
        ↪  r"C:\Users\rewan\Downloads\GP\Graduation-Project\Crowd_Detection\
33      model_path =
        ↪  r'C:\Users\rewan\Downloads\GP\Graduation-Project\Crowd_Detection\
34
35      class_list = load_class_list(class_file_path)
36      model = YOLO(model_path)
37      cap = initialize_video_capture(video_path)
38      tracker = Tracker()
39
40      if not cap.isOpened():
41          print("Error opening video capture.")
42          return
43
44      count = 0
45      while cap.isOpened():
46          ret, frame = cap.read()
47          if not ret:
48              break
49
50          count += 1
51          if count % 3 != 0:
52              continue
53
54          frame = cv2.resize(frame, (1020, 500))
55          processed_frame = detect_and_track(frame, model,
            ↪  class_list, tracker)
56          cv2.imshow("Crowd Detection", processed_frame)
57
58          if cv2.waitKey(1) & 0xFF == 27:  # Exit on 'Esc'
            ↪  key
```

```
            break

    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # i have issue when i try to import this file outside
    ↪ the folder with this line
    # so i decide to put it here till i find solution for
    ↪ it
    from tracker import Tracker
    CrowdDetector()
```

3- Visualization of Results:

Sample Output Images: Present image 1.14 showcasing the accurate detection and counting of individuals within factory environments using YOLOv8.

Figure 1.14: Result image of crowd detection model using YOLOv8, highlighting detected individuals in a crowd with bounding boxes for real-time monitoring and analysis.

This section illustrates the application of YOLOv8 for crowd detection

in factory settings, offering practical code examples and visual demonstrations that underscore its effectiveness in optimizing safety measures and operational efficiency within industrial workplaces.

Fire Detection

In this section, we explore the application of the YOLOv8 system for fire detection in industrial environments, contributing to enhanced safety and prevention measures.

1- YOLOv8 Configuration for Fire Detection:

• Model Architecture: Configuration of YOLOv8 with appropriate backbone and detection heads.

• Training Dataset: Utilization of annotated datasets containing images of fires in industrial settings.

• Training Process: Hyperparameter tuning and augmentation techniques to ensure high accuracy in detection.

2- Code Snippet:

```
def fireframe(frame, model, threshold=50):
    """Detect fire in a single frame and annotate with bounding boxes."""
    results = model(frame, stream=True, verbose=False)
    classnames = ['fire']
    fire_detected = False
    for info in results:
        boxes = info.boxes
        for box in boxes:
            confidence = box.conf[0]
            confidence = math.ceil(confidence * 100)
            Class = int(box.cls[0])
            if confidence > threshold:
                x1, y1, x2, y2 = map(int, box.xyxy[0])
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 5)
                cvzone.putTextRect(frame,
                f'{classnames[Class]} {confidence}%',
                [x1 + 8, y1 + 100],
                                   scale=3, thickness=3,
                                   colorR=(0, 0, 0))
```

```
17                  fire_detected = True
18      # Send email if fire detected and not sent within the
        ↪  last 15 minutes
19      global last_email_sent_time
20      current_time = time()
21      if fire_detected and (current_time -
        ↪  last_email_sent_time) >= 900:  # 900 seconds = 15
        ↪  minutes
22          send_email_async("WARNING: Fire detected!")
23          last_email_sent_time = current_time  # Update last
            ↪  email sent tim
24      return frame
25
26  def FirePredictor():
27      """Process video frames for fire detection."""
28      args = parse_arguments()
29      video_path = args.get("video", None)
30
31      model_path =
        ↪  r'C:\Users\rewan\Downloads\GP\B2\Fire_Detection\fire.pt'
32
33      model = YOLO(model_path)
34      cap = initialize_video_capture(video_path)
35
36      if not cap.isOpened():
37          print("Error opening video capture.")
38          return
39
40      while True:
41          ret, frame = cap.read()
42          if not ret:
43              break
44
45          processed_frame = fireframe(frame, model)
46          cv2.imshow("Fire Detection", processed_frame)
47
48          if cv2.waitKey(1) & 0xFF == 27:  # Exit on 'Esc'
            ↪  key
49              break
```

```
50
51      cap.release()
52      cv2.destroyAllWindows()
53
54  if __name__ == "__main__":
55      FirePredictor()
```

3- Visualization of Results:

Sample Output Images: Illustrate sample image 1.15 demonstrating fire detection using YOLOv8 in industrial environments.

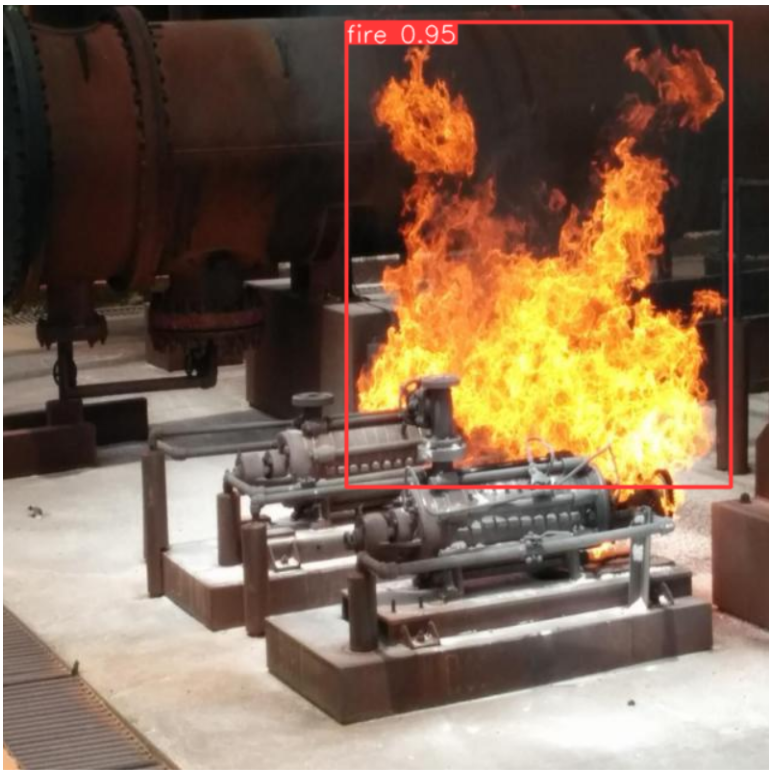

Figure 1.15: Detection of Fires Using YOLOv8: Highlighting the system's capability to swiftly identify fires in industrial environments, crucial for early intervention and enhanced workplace safety.

This approach demonstrates the implementation of a fire detection system using YOLOv8, including code examples and visual results showcasing the model's effectiveness in detecting fires efficiently in industrial settings.

Product Inspection

This section delves into the implementation of YOLOv8 for product quality inspection, emphasizing its role in maintaining high manufacturing standards and ensuring product integrity.

Product detection

Product detection using computer vision, especially with YOLO, is vital for quality inspection in manufacturing due to its: Automation: Reducing manual labor and enabling continuous operation.

- Real-time Detection: Identifying defects promptly, minimizing faulty product output.
- High Accuracy: Ensuring stringent quality standards are maintained.
- Versatility: Adapting to various product types and manufacturing environments.
- Cost-effectiveness: Leading to savings through improved efficiency and reduced scrap.
- Data-driven Decision-making: Providing valuable insights for process optimization.
- Customer Satisfaction: Enhancing trust and loyalty by delivering high-quality products consistently.

1- YOLOv8 Configuration for Product Quality Inspection:

- Model Architecture: Configuration of YOLOv8 optimized for detecting defects and anomalies in manufactured products.
- Training Dataset: Utilization of annotated datasets comprising images of products with known defects for training.
- Training Process: Fine-tuning of hyperparameters and augmentation techniques to improve detection accuracy.

2- Code Snippet:

```
# Load a model
model = YOLO(r"Model location")  # load a pretrained model (recommended for training)

# Use the model

results = model("Data location",show=True,save=True ,save_conf=True , show_labels=True ,show_conf=True)  # predict on an image
```

3- Visualization of Results:

Sample Output Images: Showcase image 1.16 illustrating the detection of product defects and anomalies using YOLOv8.

Figure 1.16: Defect Detection Using YOLOv8: Demonstrating the system's capability to accurately detect products .

This subsection demonstrates the application of YOLOv8 for product detection, providing code examples and visual results to underscore its effectiveness in maintaining manufacturing standards and ensuring product quality.

Product Counting and Tracking

Product counting and tracking utilizing YOLO in quality inspection play a pivotal role in manufacturing for various reasons:

- Accuracy: YOLO-based systems ensure precise detection and tracking, minimizing counting errors and facilitating accurate inventory management.
- Efficiency: Automation reduces reliance on manual labor, enhancing productivity and streamlining the quality inspection process.
- Real-time Monitoring: YOLO enables immediate identification of discrepancies, ensuring prompt corrective action and maintaining production integrity.
- Quality Assurance: Accurate counting and tracking uphold quality stan-

dards by ensuring the right quantity of products, mitigating risks of under- or overproduction.
• Traceability: By tracking products throughout production, YOLO aids in identifying and rectifying quality issues, ensuring product traceability and compliance.
• Preventive Maintenance: Monitoring with YOLO facilitates proactive equipment maintenance, minimizing downtime and optimizing production efficiency.
• Data Insights: YOLO-generated data offers valuable insights for process optimization and continuous improvement initiatives, enhancing operational efficiency.
• Compliance and Auditability: Accurate tracking supports regulatory compliance and provides audit trails for quality assurance, bolstering manufacturing integrity.

1- YOLOv8 Configuration for Product Counting and Tracking:

• Model Architecture: Configuration of YOLOv8 tailored for accurately counting and tracking products on assembly lines or in warehouses.
• Training Dataset: Utilization of annotated datasets containing images of products in various orientations and conditions for robust training.
• Training Process: Fine-tuning of model parameters and integration of tracking algorithms for precise counting and trajectory monitoring.

2-Code Snippet:

```
counter = object_counter.ObjectCounter()
counter.set_args( view_img=True,
                  reg_pts=line_points,
                  classes_names=model.names,
                  draw_tracks=True,
                  line_thickness=2,
                  view_in_counts = False,
                  count_reg_color= (255, 0, 0)
                          )



while cap.isOpened():
```

```
    success, im0 = cap.read()
    if not success:
        print("Video frame is empty or video processing has
        ↪  been successfully completed.")
        break
    tracks = model.track(im0, persist=True, show=False,
                         classes=classes_to_count
                         )

    im0 = counter.start_counting(im0, tracks)
    video_writer.write(im0)

cap.release()
video_writer.release()
cv2.destroyAllWindows()
```

3- Visualization of Results:

Sample Output Images: Display image 1.17 illustrating the accurate counting and tracking of products using YOLOv8.

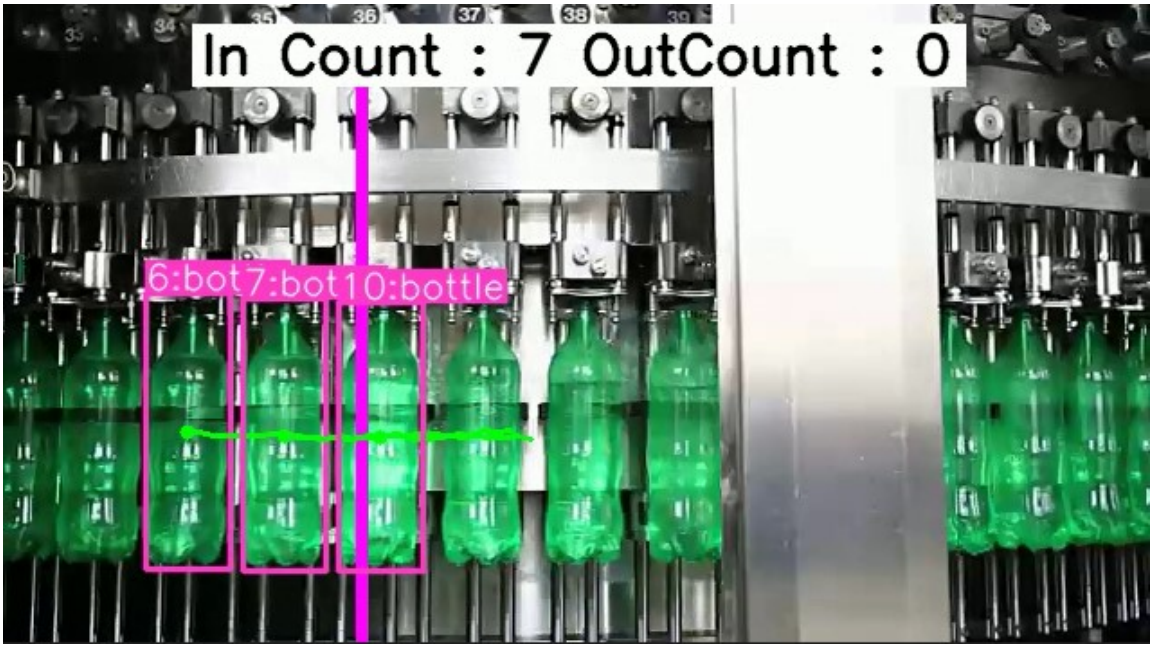

Figure 1.17: Efficient Product Counting and Tracking with YOLOv8: Demonstrating precise detection and tracking capabilities, ensuring accurate inventory management and streamlined production processes.

This subsection highlights the implementation of YOLOv8 for product counting and tracking, providing code snippets and visual outcomes that underscore its effectiveness in optimizing inventory management and enhancing manufacturing efficiency.

Defect detection and classification

Product defect detection and classification using computer vision, particularly with YOLO, holds significant importance in quality inspection for several reasons: • Early identification of defects, reducing rework costs.

- Ensuring consistent quality control and minimizing defective products.
- Cost reduction by automating manual inspection processes.
- Maintaining productivity with real-time defect detection.
- Accurate classification for targeted defect resolution.
- Data-driven decision-making for process optimization.
- Enhancing customer satisfaction and brand reputation.
- Ensuring regulatory compliance in manufacturing operations.

1- YOLOv8 Configuration for Defect Detection and Classification:

- Model Architecture: Configuration of YOLOv8 optimized for detecting and classifying various types of product defects.
- Training Dataset: Utilization of annotated datasets comprising images of defective products categorized by type of defect.
- Training Process: Fine-tuning of model parameters and integration of classification algorithms for precise defect identification.

2- Code snippet: **Defect Detection:**

```
def defectframe(frame, model):
    results = model(frame)

    annotated_frame = results[0].plot()
    return annotated_frame

if __name__ == "__main__":
    model =
     ↪  YOLO(r"C:\Users\rewan\Downloads\GP\Graduation-Project\Defects_Det

    cap = cv2.VideoCapture(0)
```

```

    if not cap.isOpened():
        print("Error: Could not open video source.")
        exit()

    while True:
        ret, frame = cap.read()

        if not ret:
            break
        annotated_frame = defectframe(frame, model)

        cv2.imshow('Defects Detection', annotated_frame)
        if cv2.waitKey(1) & 0xFF == 27:
            break

    cap.release()
    cv2.destroyAllWindows()
```

Defect classification:

```
def defectclassframe(frame, model, threshold=50):
    # Perform inference on the frame
    results = model(frame, stream=True, verbose=False)

    for info in results:
        boxes = info.boxes
        for box in boxes:
            conf = box.conf[0]
            conf = math.ceil(conf * 100)
            cls = int(box.cls[0])
            if conf >= threshold:
                x1, y1, x2, y2 = map(int, box.xyxy[0])
                label = f'{model.names[cls]} {conf:.2f}'
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0,
                ↪  255, 0), 2)
                cv2.putText(frame, label, (x1, y1 - 10),
                ↪  cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255,
                ↪  0), 2)
```

```
    return frame

if __name__ == "__main__":
    model =
    ↪ YOLO(r"C:\Users\rewan\Downloads\GP\Graduation-Project\Defects_Cla

    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        print("Error: Could not open video source.")
        exit()

    while True:
        ret, frame = cap.read()

        if not ret:
            break
        annotated_frame = defectclassframe(frame, model)

        cv2.imshow('Defects class', annotated_frame)
        if cv2.waitKey(1) & 0xFF == 27:
            break

    cap.release()
    cv2.destroyAllWindows()
```

3- Visualization of Results:

Sample Output Images: Illustrate image 1.18 showcasing the detection and classification of product defects using YOLOv8.

This subsection demonstrates the application of YOLOv8 for defect detection and classification, providing code examples and visual results that highlight its effectiveness in ensuring quality control and enhancing manufacturing processes through accurate defect identification and classification.

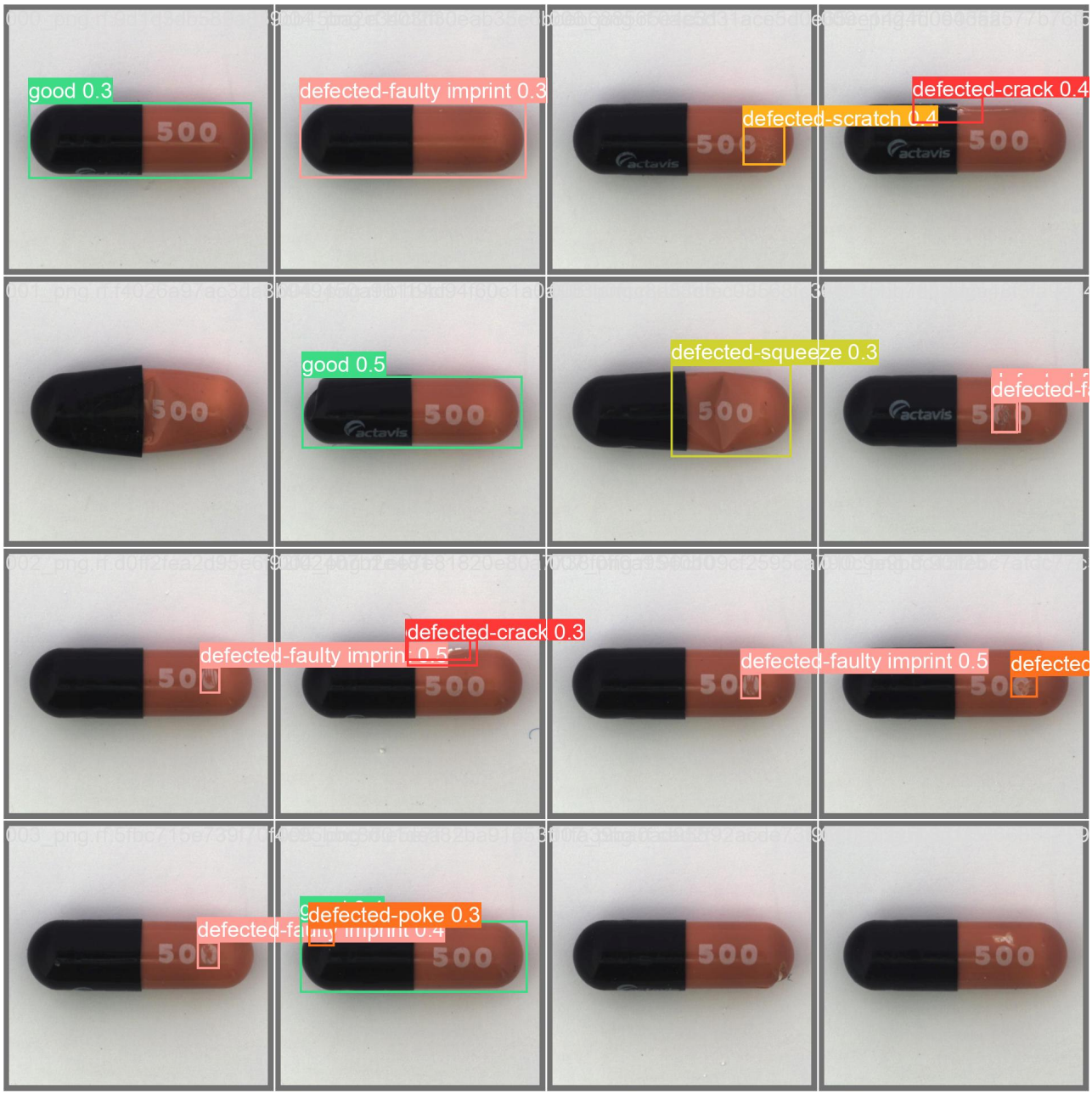

Figure 1.18: Precision in Defect Detection and Classification with YOLOv8: Highlighting the system's ability to identify and categorize product defects, ensuring stringent quality control in manufacturing.

Barcode Recognition

This chapter explores the implementation of YOLOv8 for barcode recognition, emphasizing its role in enhancing identification processes for products and workers in manufacturing and logistics environments. Real-time barcode detection and products and workers recognition using YOLO object detection and PyZBar. Includes image and video processing scripts for

identifying products and workers from barcodes.

Features

• Object Detection: Utilizes YOLO (You Only Look Once) for accurate and efficient object localization.
• Barcode Decoding: Uses PyZBar library to decode barcode information from cropped regions of interest in images or video frames.
• Product and worker Recognition: Matches decoded barcode information against a predefined product or worker database to identify them.
• Real-time Processing: Supports live video processing for instant recognition.

1- YOLOv8 Configuration for Barcode Recognition:

• Model Architecture: Configuration of YOLOv8 optimized for accurately detecting and decoding barcodes on products and worker badges.
• Training Dataset: Utilization of annotated datasets containing images of various types of barcodes and their placements.
• Training Process: Fine-tuning of model parameters and integration of decoding algorithms for precise barcode recognition.

2- Code snippet:

Image Predict:

```
def ImagePredictor(image_path):
    # Load YOLO model
    model_path = os.path.join('.', 'last.pt')
    model = YOLO(model_path)

    # Set detection threshold
    threshold = 0.5

    # Read input image
    img = cv2.imread(image_path)

    # Predict using YOLO
    results = model(image_path)[0] # predict on an image

```

```
    for result in results.boxes.data.tolist():
        x1, y1, x2, y2, score, class_id = result
        # Crop the detected region
        croped_img = img[int(y1)-50:int(y2)+50,
        ↪  int(x1)-50:int(x2)+50]
        text = " "
        try:
            # Decode barcode from the cropped image
            barcode = decode(croped_img)[0]
            barcode = barcode.data.decode('utf-8')

            # Check if the barcode exists in the product
            ↪  database
            with open(os.path.join('.', 'products.json'),
            ↪  "r") as products:
                data = json.load(products)
                if barcode in data:
                    text = data[barcode]
        except:
            print("Can't be decoded")

        if score > threshold:
            # Draw bounding box and display product name
            cv2.rectangle(img, (int(x1), int(y1)), (int(x2),
            ↪  int(y2)), (0, 255, 0), 4)
            cv2.putText(img, text, (int(x1), int(y1 - 10)),
                        cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0,
                        ↪  255, 0), 3, cv2.LINE_AA)
    # Display the processed image
    cv2.imshow('Image', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Parse command-line arguments
    parser = argparse.ArgumentParser()
    parser.add_argument('-i','--image', help='path to the
    ↪  input image file')
    args = vars(parser.parse_args())
```

```
48
49      # Call main function with the specified image path
50      ImagePredictor(args["image"])
```

Barcode Predict:

```
1   def Barcodeframe(frame, model, threshold=50):
2       """Perform object detection and barcode decoding on a
        ↪  single frame."""
3       results = model(frame, verbose=False)[0]
4
5       for result in results.boxes.data.tolist():
6           x1, y1, x2, y2, score, class_id = result
7           cropped_frame = frame[int(y1)-50:int(y2)+50,
            ↪  int(x1)-50:int(x2)+50]
8           text = " "
9
10          try:
11              barcode = decode(cropped_frame)[0]
12              barcode = barcode.data.decode('utf-8')
13              text = barcode
14              with open('products.json', "r") as products:
15                  data = json.load(products)
16                  text = data[barcode]
17          except Exception as e:
18              #print(f"Error decoding barcode: {e}")
19              pass
20
21          if score > (threshold/100):
22              cv2.rectangle(frame, (int(x1), int(y1)),
                ↪  (int(x2), int(y2)), (0, 255, 0), 4)
23              cv2.putText(frame, text.upper(), (int(x1),
                ↪  int(y1 - 10)),
24                          cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0,
                            ↪  0, 0), 3, cv2.LINE_AA)
25
26      return frame
27
28  def VideoPredictor():
```

```
    """Main function to process video frames."""
    args = parse_arguments()
    video_path = args.get("video", None)

    model_path =
    ↪ r"C:\Users\rewan\Downloads\GP\Graduation-Project\Barcode_Product_
    model = YOLO(model_path)

    cap = initialize_video_capture(video_path)
    if not cap.isOpened():
        print("Error opening video capture.")
        return

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        processed_frame = Barcodeframe(frame, model)
        cv2.imshow("Frame", processed_frame)

        if cv2.waitKey(1) & 0xFF == 27:  # Exit on 'Esc'
        ↪ key
            break

    cap.release()
    cv2.destroyAllWindows()

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

3- Visualization of Results:

Sample Output Images: Showcase images 1.19 and 1.20 demonstrating the accurate recognition and decoding of barcodes using YOLOv8 for products and worker badges.

This section illustrates the application of YOLOv8 for barcode recognition, providing practical code snippets and visual examples that highlight its effectiveness in improving identification processes and operational effi-

Figure 1.19: Accurate Product Barcode Detection with YOLOv8: Ensuring seamless inventory management and operational efficiency in manufacturing and logistics.

Figure 1.20: Worker Barcode Detection with YOLOv8: Demonstrating precise recognition capabilities for enhancing workplace identification and operational efficiency.

ciency across manufacturing and logistics sectors.

Chapter 2

Embedded System

2.1 Why STM32...?

STM32 microcontrollers are a workhorse for building Internet of Things (IoT) devices due to their strong combination of features and affordability. Their processing power allows them to not only collect data from various sensors (temperature, light, pressure) but also perform onboard data analysis and control functions. This reduces reliance on external processing units and lowers overall project complexity. The wide range of STM32 models with varying processing power, memory capacity, and connectivity options (WiFi, Bluetooth, cellular) empowers developers to choose the perfect fit for their project's needs. Additionally, STMicroelectronics, the manufacturer of STM32, provides a robust development ecosystem with software libraries, online tools, and a large community for support, making it easier for beginners and experienced developers alike to create innovative IoT applications.

Additionally, STMicroelectronics offers specific STM32 lines that boast built-in LoRa support, a Low Power Wide Area Network (LPWAN) technology ideal for IoT applications requiring long-range communication with low power consumption. This makes STM32 microcontrollers suitable for projects like remote environmental monitoring, asset tracking, and industrial automation in geographically dispersed locations. LoRa technology enables sending small data packets over extended distances, even in rural areas, while preserving battery life for battery-powered devices – a critical factor for many IoT applications. The combination of STM32's processing power, versatile communication options, and LoRa support empowers developers to create feature-rich IoT devices that operate efficiently over long ranges. Stepping into the realm of Long Range communication for IoT with STM32, specific lines like the STM32WL series come equipped

with integrated LoRa support. LoRa, short for Long Range, is a Low Power Wide Area Network (LPWAN) technology that shines in applications demanding long-distance data transmission with minimal power consumption. This makes STM32 with LoRa a perfect fit for projects like:

- Remote Environmental Monitoring:Imagine a network of sensors scattered across a vast forest, monitoring temperature, humidity, and soil moisture. Traditional communication protocols might struggle with the range and battery life needed. STM32 with LoRa allows these sensors to send small data packets over significant distances, even in rural areas with limited infrastructure. This enables researchers to collect vital environmental data from geographically dispersed locations.

- Asset Tracking: For companies managing a fleet of vehicles or valuable equipment across a wide area, real-time location tracking is crucial. LoRa-enabled STM32 microcontrollers can be embedded within these assets, periodically transmitting location data. This allows for efficient monitoring of asset movement, optimizing logistics and preventing theft.

- Industrial Automation in Spread-Out Facilities: Large industrial facilities often have sensors and actuators spread across a significant area. With traditional protocols, cabling for communication can become complex and expensive. STM32 with LoRa offers a wireless solution, enabling these devices to communicate efficiently over long distances within the facility, facilitating remote monitoring and control of industrial processes.

The key advantage of LoRa lies in its ability to strike a balance between communication range and power consumption. Unlike traditional cellular networks that require significant power to maintain constant connection, LoRa utilizes a spread spectrum technique to transmit small data packets over extended distances. This translates to lower power consumption, allowing battery-powered devices equipped with STM32 and LoRa to operate for extended periods without frequent recharging – a critical factor for many remote IoT applications.

In conclusion, the marriage of STM32's processing power and LoRa technology unlocks a new level of capabilities for developers building long-range, low-power IoT solutions. This empowers them to create feature-rich

devices that operate efficiently over vast distances, paving the way for innovative applications in remote environmental monitoring, asset tracking, industrial automation, and various other scenarios demanding long-range and power-conscious communication. We will Talk about lora in seperate chapter.

2.2 STM32 in our project

For our graduation project we have used STM32F103C8 ,since The STM32F103C8 belongs to the STM32F1 series, which is one of the most available and popular microcontroller families for several reasons:

- **Widespread Availability** - STMicroelectronics, the manufacturer, produces the STM32 in high volumes, making them readily available from numerous distributors worldwide. You can easily find them online or through electronic component stores. This widespread availability makes them a reliable choice for projects where you don't want to be restricted by limited component options.
- **Affordability** - Compared to some high-performance microcontrollers, the STM32F1 series offers excellent value for the features it provides. This affordability makes it an ideal choice for student projects where budget might be a constraint.
- **Versatility** - The STM32F1 series boasts a wide range of models with varying processing power, memory capacity, and peripheral options. This allows you to choose the exact model that best suits your project's needs without having to pay for unnecessary features. The STM32F103C8 itself is a good example, offering a good balance of performance and affordability for various projects requiring moderate computing power.
- **Large User Community** - Due to its popularity, the STM32F1 series has a vast and active user community. This translates to a wealth of online resources like tutorials, sample code, forums, and project examples. If you encounter any challenges during your project, you're likely to find solutions or helpful discussions from other users who have tackled similar issues
- **Maturity and Stability** - The STM32F1 series has been around for a considerable time, making it a mature and well-established platform. This maturity translates to stable development tools, reliable hardware performance, and a lower risk of encountering unexpected bugs or compatibility issues.

2.3 STM32F103C8T6

The STM32F103C8T6 is a microcontroller from STMicroelectronics' STM32 family. Here's a detailed breakdown of what can be deduced from its name and the associated features:

Breakdown of STM32F103C8T6

- **STM32**: Indicates that it belongs to the STM32 family of microcontrollers, which are based on ARM Cortex-M cores.
- **F1**: Denotes the series within the STM32 family. The F1 series is known for its balance between performance and power efficiency and is based on the ARM Cortex-M3 core.
- **03**: Indicates the line within the F1 series. The "03" line is typically targeted for mainstream applications.
- **C**: Specifies the package type. "C" usually represents a 48-pin package (LQFP48)[1].

  .
- **8**: Indicates the flash memory size. "8" means the microcontroller has 64 KB of flash memory.
- **T6**: Refers to the temperature range and the package type:
  - **T**: Indicates the operating temperature range, which is typically -40 to 85°C for industrial applications.
  - **6**: Indicates the specific type of packaging, which in this case is LQFP (Low-Profile Quad Flat Package).

Key Features of STM32F103C8T6

- **Core**: ARM Cortex-M3, a 32-bit RISC processor core designed for high performance and low power consumption.
- **Flash Memory**: 64 KB, used for storing the program code.
- **SRAM**: 20 KB, for data storage and manipulation during program execution.

[1]LQFP stands for Low-Profile Quad Flat Package, which is a surface-mount integrated circuit package with a flat, quad-shaped design and reduced height.

- **Clock Speed**: Up to 72 MHz, providing a good balance between performance and power consumption.
- **Peripherals**:
  - **GPIOs**: General-purpose input/output pins for interfacing with external devices.
  - **Timers**: Multiple timers for tasks such as generating PWM signals, measuring input pulse widths, and creating delays.
  - **USART/UART**: Communication interfaces for serial communication.
  - **SPI**: Serial Peripheral Interface for communication with peripherals like sensors, displays, and memory devices.
  - **I2C**: Inter-Integrated Circuit interface for communication with other microcontrollers and peripherals.
  - **ADC**: Analog-to-Digital Converters for reading analog signals from sensors.
  - **USB**: USB 2.0 full-speed interface for connecting to USB devices.

Application in Projects

The STM32F103C8T6 is widely used in various applications, including:

- **Embedded Systems**: For control and monitoring applications.
- **IoT Devices**: Due to its balance of power and performance.
- **Industrial Automation**: For tasks like motor control, sensor data acquisition, and communication.
- **Consumer Electronics**: In products requiring user interfaces, data logging, and communication capabilities.

Example Project: YOLOv8 for Crowd Detection

If you're using the STM32F103C8T6 in a project that involves crowd detection using YOLOv8, the microcontroller could play a role in interfacing with sensors, managing communications, and controlling actuators based on the detection results. For instance:

- **Sensor Interface**: Connecting to cameras or other sensors to acquire data.
- **Communication**: Using LoRa, UART, or USB to send data to a central server or other devices.
- **Control Logic**: Implementing logic to respond to detected crowds, such as activating alarms or controlling access.

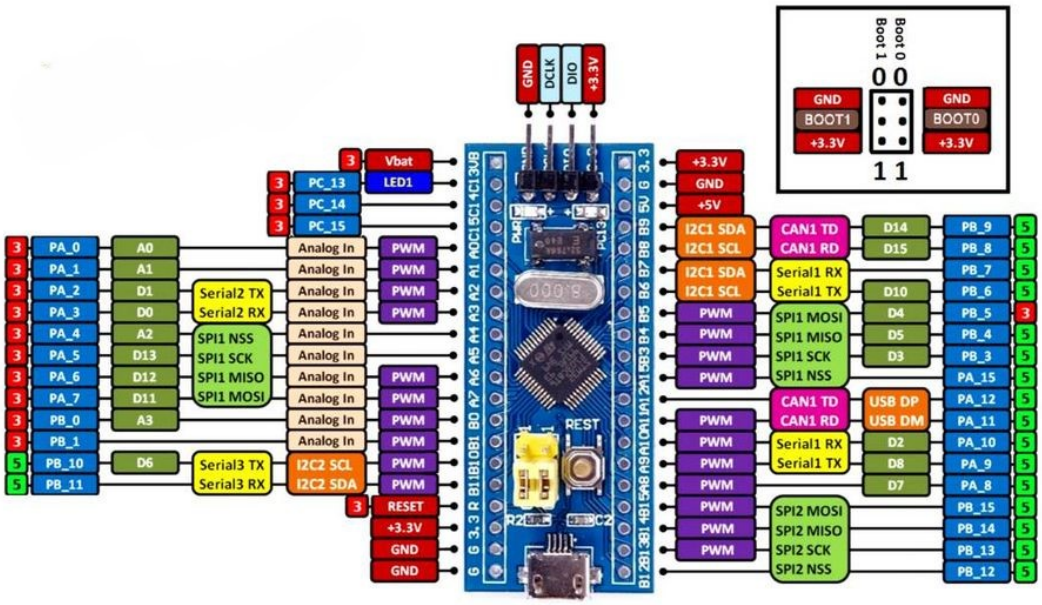

Figure 2.1: STM32F103C8T6

2.4 Wireless Sensor Data Transmission Using STM32 and ESP32 with LoRa

Project Overview

This project involves transmitting sensor data wirelessly from an STM32F103C8T6 microcontroller (acting as the transmitter) to an ESP32 microcontroller (acting as the receiver) using LoRa communication. The ESP32 then sends the received data to a PC, where it is displayed on a desktop application.

Components

- **Microcontrollers**:
  - **STM32F103C8T6**: A low-power ARM Cortex-M3 MCU used for reading sensor data and transmitting it.
  - **ESP32**: A versatile MCU with built-in Wi-Fi and Bluetooth, used for receiving data and communicating with the PC.
- **LoRa Modules**:
  - **SX1278**: Long Range (LoRa) transceivers used for low-power, long-range wireless communication.
- **Sensors**:
  - **DHT11**: Temperature and humidity sensor.
  - **MQ4**: Gas sensor.
- **PC**: Receives data from the ESP32 via serial communication and displays it on a desktop application.

Architecture

- **Transmitter (STM32F103C8T6 + LoRa SX1278)**:
  - **Sensor Interface**: Code to initialize and read data from sensors.
  - **LoRa Communication**: Code to initialize and transmit data using the SX1278 module.
- **Receiver (ESP32 + LoRa SX1278)**:
  - **LoRa Communication**: Code to initialize and receive data from the SX1278 module.
  - **Serial Communication**: Code to send received data to the PC via UART or USB.
- **PC Application**:
  - **Serial Communication**: Code to read data from the ESP32.
  - **Data Display**: Code to display the received data on the GUI of the desktop application.

Implementation Steps

Transmitter (STM32F103C8T6)

1. **Setup Sensors**:
    - Initialize the DHT11 and MQ4 sensors and read data from them.
2. **LoRa Transmission**:
    - Initialize the SX1278 module for LoRa communication.
    - Transmit the sensor data using LoRa.

Listing 2.1: STM32 HAL Library Code Example

```
#include "main.h"
#include "sx1278.h"  // Assume a library for SX1278 exists
#include "dht11.h"   // Assume a library for DHT11 exists
#include "mq4.h"     // Assume a library for MQ4 exists

void setup() {
    // Initialize sensors
    DHT11_Init();
    MQ4_Init();
    // Initialize LoRa module (SX1278)
    SX1278_Init();
}

void loop() {
    // Read sensor data
    char sensor_data[100];
    float temperature = DHT11_ReadTemperature();
    float humidity = DHT11_ReadHumidity();
    int gas_level = MQ4_Read();

    // Format sensor data into a string
    sprintf(sensor_data, "Temp: %.2f C, Hum: %.2f %, Gas: %d",

    // Transmit data via LoRa
    SX1278_Transmit(sensor_data, strlen(sensor_data));
```

```
    // Delay before next transmission
    HAL_Delay(1000);
}
```

Receiver (ESP32)

1. **LoRa Reception**:
    - Initialize the SX1278 module for receiving data.
2. **Serial Communication**:
    - Initialize UART or USB communication with the PC.
    - Send the received data to the PC.

Listing 2.2: ESP32 with Arduino Framework Code Example

```
#include <SPI.h>
#include <LoRa.h>

void setup() {
    Serial.begin(9600); // Initialize serial communication
    LoRa.begin(433E6);  // Initialize LoRa at 433 MHz
}

void loop() {
    int packetSize = LoRa.parsePacket();
    if (packetSize) {
        String receivedData = "";
        while (LoRa.available()) {
            receivedData += (char)LoRa.read();
        }
        Serial.println(receivedData); // Send data to PC via S
    }
}
```

PC Application

1. **Serial Communication**:
    - Use a library like `pyserial` in Python to read data from the serial port connected to the ESP32.

2. **Data Display**:
    - Display the received data on a graphical user interface (GUI).

Listing 2.3: Python with `pyserial` Code Example

```
import serial
import time

# Initialize serial communication with the ESP32
ser = serial.Serial('COM3', 9600)  # Change 'COM3' to the appro
time.sleep(2)  # Wait for the serial connection to initialize

while True:
    if ser.in_waiting > 0:
        line = ser.readline().decode('utf-8').rstrip()
        print(line)  # Display the received data
        # Here you can add code to update the GUI with the rece
```

Conclusion

This project involves transmitting sensor data from an STM32F103C8T6 microcontroller to an ESP32 microcontroller using LoRa SX1278 modules. The ESP32 then sends the data to a PC, where it is displayed on a desktop application. By following the outlined steps and using the provided code examples, you can successfully implement this wireless communication system and visualize the sensor data on a PC.

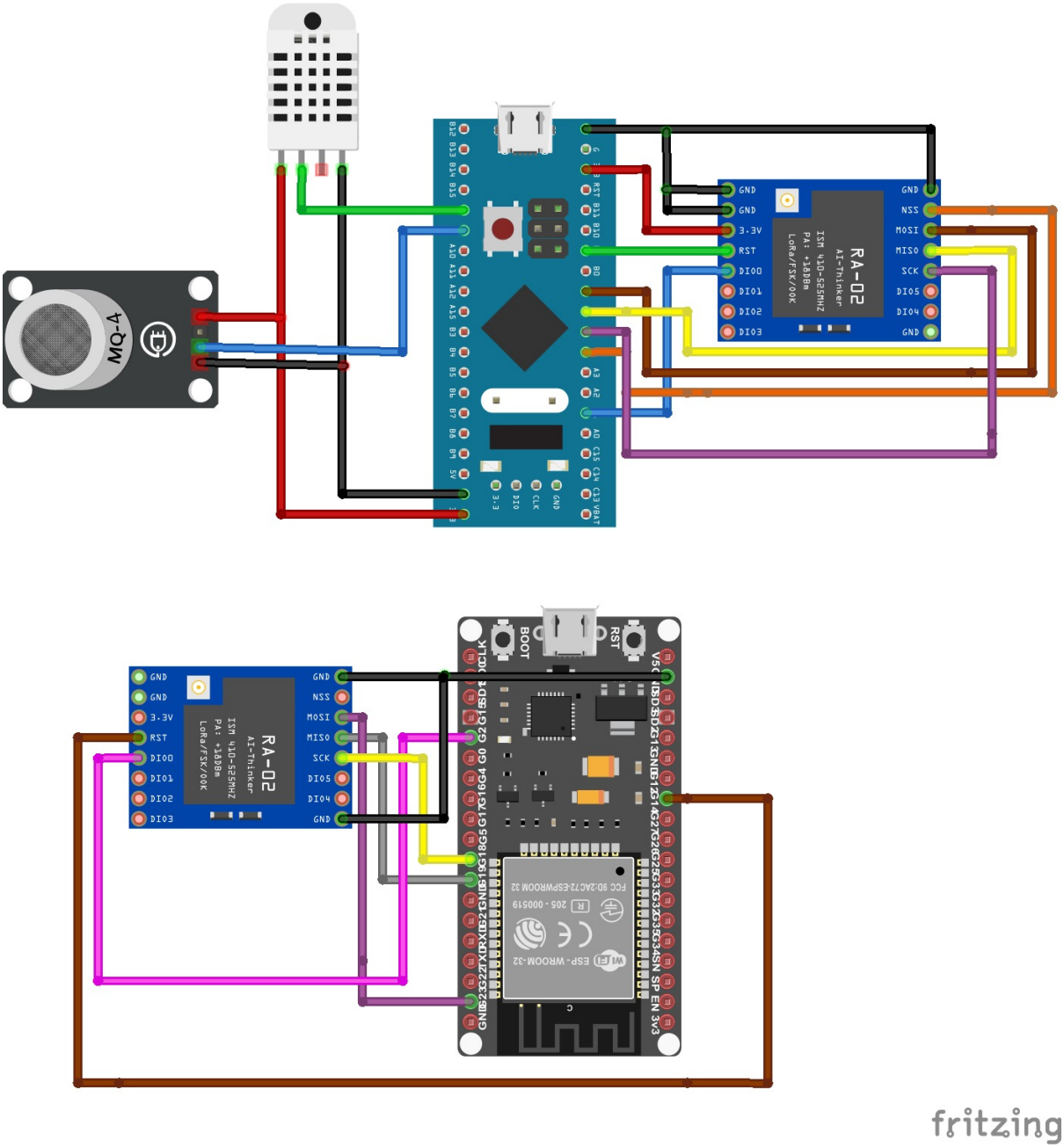

Figure 2.2: Wireless Sensor Data Transmission Using STM32 and ESP32 with LoRa

2.5 Received Packet Analysis During Fire Test

During the intentional fire test, the received packets likely captured the following data:

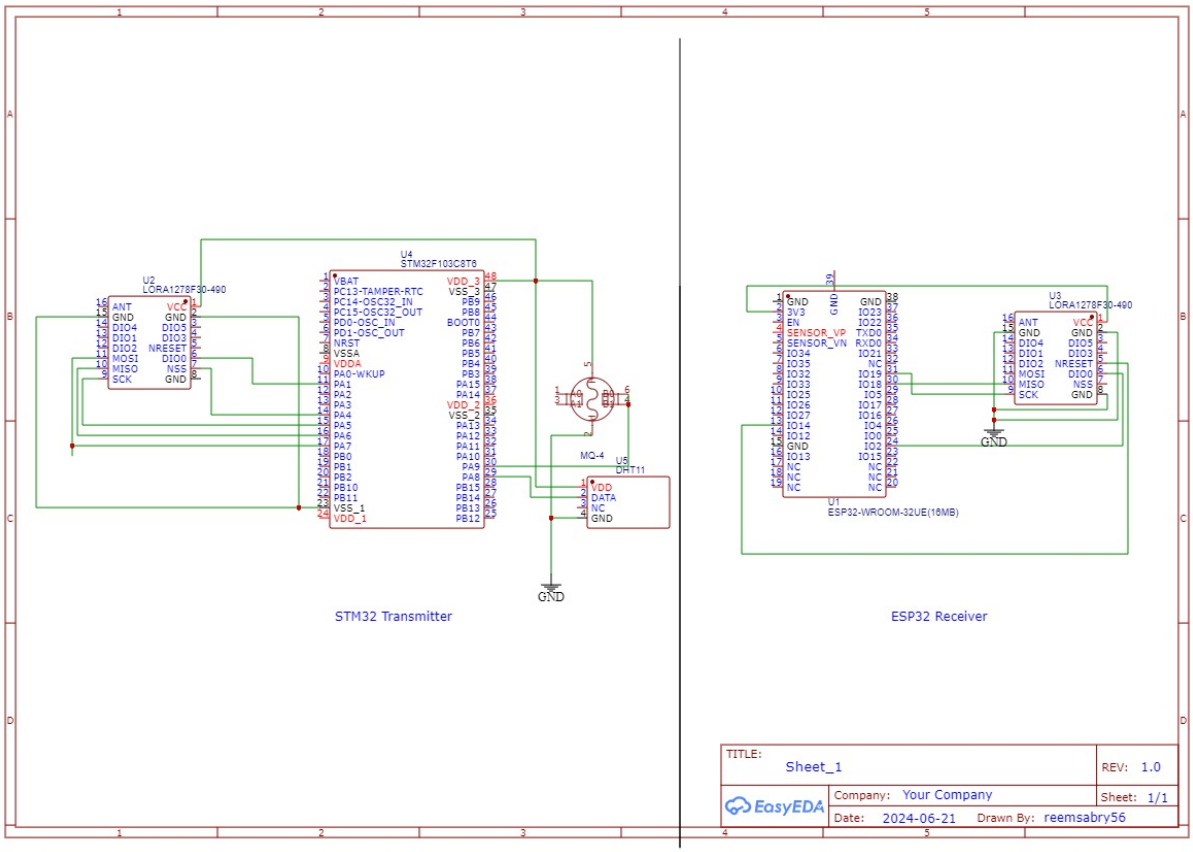

Figure 2.3: Wireless Sensor Data Transmission Using STM32 and ESP32 with LoRa

Temperature and Humidity

These values may vary slightly due to the environmental changes caused by the fire, but typically, the temperature would rise while humidity might decrease.

Gas Readings

- **LPG (Liquefied Petroleum Gas):** This should increase significantly during the fire due to the combustion of hydrocarbons.
- **CH4 (Methane):** Similar to LPG, methane levels would rise due to the burning of organic materials.
- **CO (Carbon Monoxide):** CO levels would also increase substantially as a byproduct of incomplete combustion.
- **Alcohol:** Ethanol levels may increase due to the combustion of alcohol-based fuels or other flammable liquids.

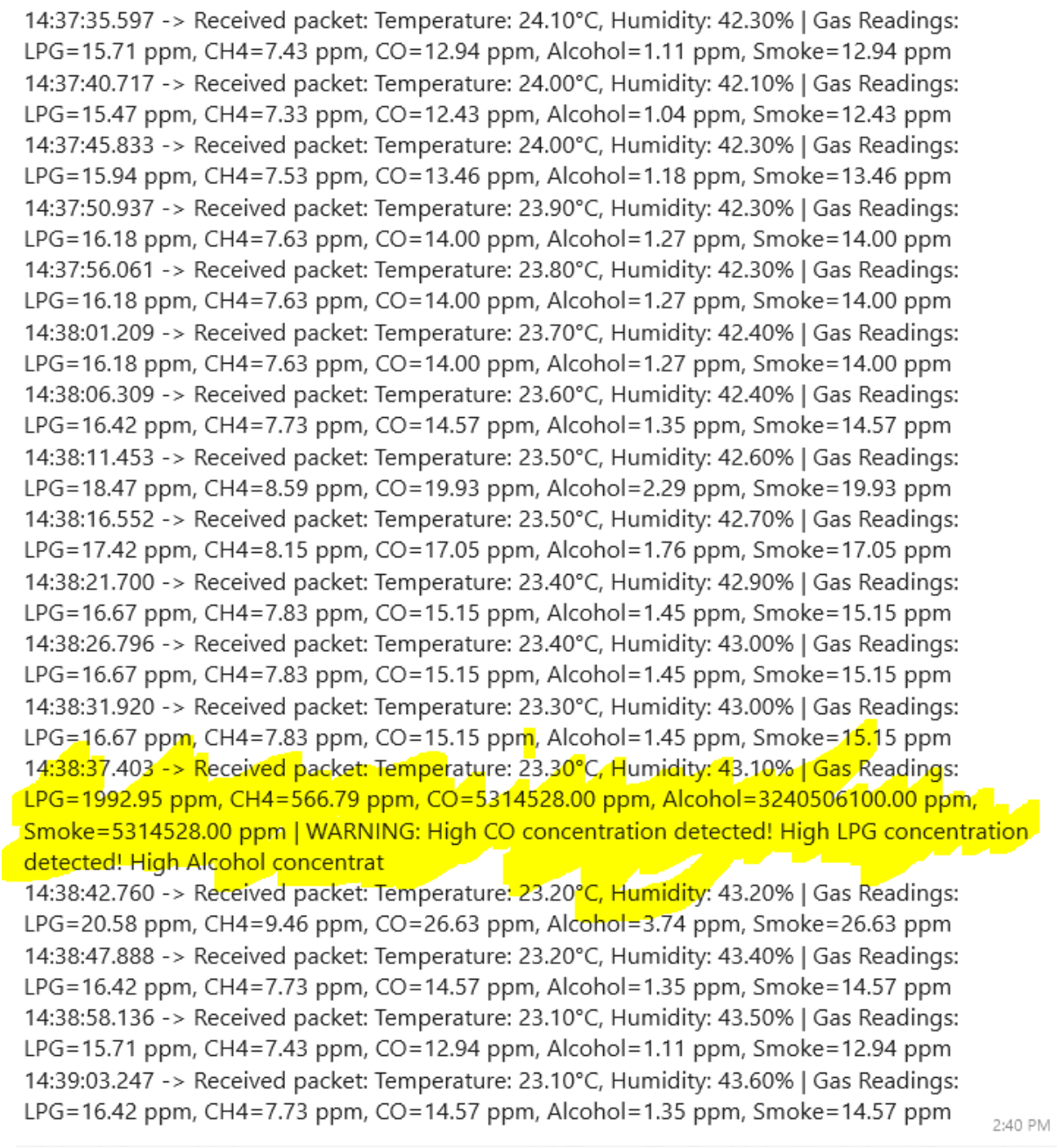

Figure 2.4: Enter Caption

- **Smoke:** Smoke particles would significantly increase as visible evidence of combustion.

Interpretation of PPM (Parts Per Million) Units

PPM (Parts Per Million) denotes the concentration of a substance in the air relative to the total air volume.

Expected Behavior During the Test

Temperature and Humidity

Minor variations may occur due to environmental changes caused by the fire, affecting the sensor readings.

Gas Concentrations

- **LPG and CH4:** Significant increases in these gases would indicate the presence of burning hydrocarbons.
- **CO:** A sharp increase in carbon monoxide levels indicates incomplete combustion and poses a serious health hazard.
- **Alcohol:** Depending on the fuel used in the fire, ethanol levels may increase.
- **Smoke:** A substantial increase in smoke particles would be expected as a visible byproduct of combustion.

Handling of Spikes and Abnormal Readings

Normal Readings

During the fire test, spikes in gas concentrations (LPG, CH4, CO) are expected and indicate the sensor's response to the fire.

Erroneous Readings

If readings such as CO at 5314528.00 ppm or similar very high values were observed during the fire test, these would likely be indicative of sensor saturation or malfunction rather than actual concentrations. Realistic readings are usually lower but significantly elevated during fire tests.

Conclusion

During the fire test, the spikes observed in the gas readings (LPG, CH4, CO) and smoke indicate the sensors' response to the intentional combustion. Such spikes are expected and demonstrate the sensors' capability to detect and respond to changes in environmental conditions, particularly those associated with fire and combustion processes. It's crucial to

ensure that sensor readings, especially during abnormal spikes, are interpreted within the context of the test scenario to differentiate between actual environmental changes and sensor anomalies. Implementing robust error-handling and validation mechanisms will help ensure the reliability of sensor data in real-world applications.

Chapter 3

LORA

3.1 Introduction

Recently new transceiver technologies have emerged which enable power efficient communication over very long distances. Examples of such LP-WAN technologies are LoRa, Sigfox and Weightless. These new transceiver types target applications where thousands of devices are used in a large geographic area to collect sensor readings. A typical application is the collection of meter readings in a city. These systems are used in a setup where simple devices send data in one hop to powerful receiver which then forwards data over a fixed wired infrastructure to a data collection point. We argue that these transceivers are potentially very useful to construct more generic IoT networks incorporating multi-hop bi-directional communications enabling sensing and actuation. The transceivers have the ability to communicate over large distances on a small energy budget which would enable us to build more efficient IoT infrastructures than currently possible. For example, commonly used ZigBee transceivers such as the Chipcon CC2420 cover a communication range of 20 m using 84.5 µJ (40 byte message) in a built up environment. A LoRa Semtech SX1272 transceiver can cover a distance of 150 m using 86.5 mJ in the same environment.
Besides improved communication range, the transceivers have unique features stemming from the used modulation schemes. Thus, it is not efficient to simply use these transceivers with existing Medium Access Control (MAC) protocols and routing mechanisms that have emerged in the IoT domain. When construction a network using these transceivers their specific capabilities should be taken into account to maximise performance in terms of communication and minimise energy consumption.
In this chapter we investigate LoRa as technology for building generic IoT networks. We investigate the communication capability of the LoRa Semtech SX1272 transceiver and its energy consumption patterns. We

analyse in detail unique communication features offered by the transceiver. A detailed analysis of these features is essential as they should be exploited when constructing communication protocols on top of this communication technology. Finally we construct an example communication protocol for the LoRa physical layer which enables wide-area multi-hop data collection and actuation without existing backbone infrastructure. Our deployment experiment demonstrates that 6 LoRa nodes can form a network covering 1.5 ha in a built up environment, achieving a potential lifetime of 2 years on 2 AA batteries and delivering data within 5 s and reliability of 80%.

3.2 LoRa

LoRa (Long Range) is a wireless communication technology specifically designed for long-range, low-power, and low-data-rate applications. It utilizes a proprietary modulation scheme called Chirp Spread Spectrum (CSS) to achieve long-range communication while maintaining low power consumption. LoRa is a key technology for the Internet of Things (IoT), enabling devices to communicate over several kilometers while consuming minimal power.

Key Features
- Long Range: LoRa technology can achieve communication distances ranging from a few kilometers in urban areas to over 15 kilometers in rural settings.
- Low Power Consumption: Devices using LoRa technology can operate on batteries for several years, making it ideal for remote and battery-powered applications.
- Low Data Rate: LoRa is optimized for applications that require low data rates, typically ranging from 0.3 kbps to 50 kbps.
- Robustness: The CSS modulation technique used by LoRa provides resilience against interference and multipath fading, enhancing reliability in challenging environments.

The development of LoRa technology can be traced back to the early 2000s, with significant milestones marking its evolution:

Early Development
- Semtech Corporation: The foundation of LoRa technology was laid by Semtech Corporation, a leading supplier of analog and mixed-signal semi-

conductors. In 2012, Semtech acquired the French company Cycleo, which had developed the underlying CSS modulation technology that would become the basis for LoRa.

- CSS Modulation: The Chirp Spread Spectrum modulation technique, originally used in military and space communication due to its robustness and long-range capabilities, was adapted for low-power wide-area network (LPWAN) applications.

LoRaWAN

As LoRa is capable to transmit over very long distances it was decided that LoRaWAN only needs to support a star topology. Nodes transmit directly to a gateway which is powered and connected to a backbone infrastructure. Gateways are powerful devices with powerful radios capable to receive and decode multiple concurrent transmissions (up to 50).
Three classes of node devices are defined:
(1) Class A enddevices: The node transmits to the gateway when needed. After transmission the node opens a receive window to obtain queued messages from the gateway.
(2) Class B enddevices with scheduled receive slots: The node behaves like a Class A node with additional receive windows at scheduled times. Gateway beacons are used for time synchronisation of end-devices.
(3) Class C end-devices with maximal receive slots: these nodes are continuous listening which makes them unsuitable for battery powered operations.

In this chapter we propose an alternative MAC for LoRa which enables multihop communication in a network of battery operated andduty-cycled devices. Although star networks with a powered and powerful gateway device are an option in some situations it does not cover all IoT application scenarios.

LoRa Physical Layer

LoRa is a physical layer specification based on CSS with integrated Forward Error Correction (FEC). Transmissions use a wide band to counter interference and to handle frequency offsets due to low cost crystals. A LoRa receiver can decode transmissions 19.5 dB below the noise floor.

Thus, very long communication distances can be bridged. LoRa key properties are: long range, high robustness, multipath resistance, Doppler resistance, low power. LoRa operates in the lower ISM bands (EU: 868 MHz and 433 MHz, USA:915 MHz and 433 MHz).

A LoRa radio has four configuration parameters: carrier frequency, spreading factor, bandwidth and coding rate. The selection of these parameters determines energy consumption, transmission range and resilience to noise. In the following sections we use the Semtech SX1272 transceiver as reference point.

Carrier Frequency
Carrier Frequency (CF) is the centre frequency used for the transmission band. For the SX1272 it is in the range of 860 MHz to 1020 MHz, programmable in steps of 61 Hz. The alternative radio chip Semtech SX1276 allows adjustment from 137 MHz to 1020 MHz.

Spreading Factor
SF is the ratio between the symbol rate and chip rate. A higher spreading factor increases the Signal to Noise Ratio (SNR), and thus sensitivity and range, but also increases the air time of the packet. The number of chips per symbol is calculated as 2sf. For example, with an SF of 12 (SF12) 4096 chips/symbol are used. Each increase in SF halves the transmission rate and, hence, doubles transmission duration and ultimately energy consumption. Spreading factor can be selected from 6 to 12. SF6, with the highest rate transmission, is a special case and requires special operations. For example, implicit headers are required. Radio communications with different SF are orthogonal to each other and network separation using different SF is possible.

Bandwidth
Bandwidth (BW) is the range of frequencies in the transmission band. Higher BW gives a higher data rate (thus shorter time on air), but a lower sensitivity (due to integration of additional noise). A lower BW gives a higher sensitivity, but a lower data rate. Lower BW also requires more accurate crystals (less ppm). Data is send out at a chip rate equal to the bandwidth. So, a bandwidth of 125 kHz corresponds to a chip rate of 125 kcps. The SX1272 has three programmable bandwidth settings: 500 kHz, 250 kHz and 125 kHz. The Semtech SX1272 can be programmed in the range of 7.8 kHz to 500 kHz, though bandwidths lower than 62.5 kHz

Figure 3.1: SX1278

requires a temperature compensated crystal oscillator (TCXO).

Coding Rate

Coding Rate (CR) is the FEC rate used by the LoRa modem and offers protection against bursts of interference. A higher CR offers more protection, but increases time on air. Radios with different CR (and same CF/SF/BW), can still communicate with each other. CR of the payload is stored in the header of the packet, which is always encoded at 4/8.

LoRa Characteristics

Using a LoRa radio in a sensor network has some interesting aspects. First, since the range is relatively large (hundreds of meter indoors, kilometres outdoors), networks can span large areas without routing over many hops. In many cases one hop from every node to the sink is feasible. Secondly, transmission on the same carrier frequency, but with different spreading factor, are orthogonal. This creates the opportunity of dividing the channel in virtual subchannels. Thirdly, when transmissions occur at the same time with the same parameters, the strongest transmission will be received with high probability. concurrent transmissions are nondestructive even when their contents is different. This feature is exploited by LoRaWAN where all gateways broadcast beacons at the same time (tight clock synchronisation via GPS) and an end device is able to demodulate the strongest beacon.

3.3 LoRa Feature Evaluation

LoRa has interesting features aside the increased communication range that should be taken into account when constructing network protocols. For example, channel separation using different SF is possible, concurrent nondestructive transmissions are possible and carrier detection via CAD is provided. However, from available documentation the performance and ability of these features is not clear. Therefore we carry out a series of experiments to evaluate these provided features.

Spreading Factors

Different spreading factors are claimed to be orthogonal to each other. Thus, construction of virtual channels on the same carrier frequency is possible (Code Division Multiple Access (CDMA)).
We evaluate how well this separation works using a simple experiment setup. A transmitter is set to continuously send a 40 B packet with a fixed SF. A receiver set to the same SF is used to receive the transmissions. A second transmitter is used to transmit continuously and sequentially using all other SF to the same receiver.
Findings
All transmissions where sender and receiver use the same SF are received correctly. None of the transmissions emitted by the second node using a different SF are received. This result suggests that channel separation using SF works perfectly.
However, as we will show in Section 4.3 this is only partially true. When using CAD to detect an incoming transmission a signal using the wrong SF may be detected as valid transmission even though it cannot be decoded. This is important as the false detection rate has a negative impact on energy efficiency of a protocol.

Concurrent Transmissions

In LoRa concurrent transmissions are claimed to be nondestructive and such feature is very valuable for protocol design. Well-timed cooperative transmissions have been used in Glossy. In Glossy the same message is transmitted accurately timed by multiple nodes allowing correct reception. A-MAC and also Whitehouse et al. make use of the capture effect. Here multiple different messages are transmitted concurrently and depending

on power levels and timing one of the concurrently transmitted messages can be received.
We set up an experiment to understand the exact conditions in which this effect is present in LoRa. We use a receiver, one 'weak' transmitter and one 'strong' transmitter (1 dBm difference). Both transmitters send a packet with explicit header and CRC. The strong transmitter varied the transmission time offset relative to the weak transmitter. From being one packet (airtime) early to being one packet (airtime) late. For each offset, 16 32-byte packets were transmitted using first identical packet payloads and subsequently different payloads. We also run the experiment with all combinations of SF and BW.

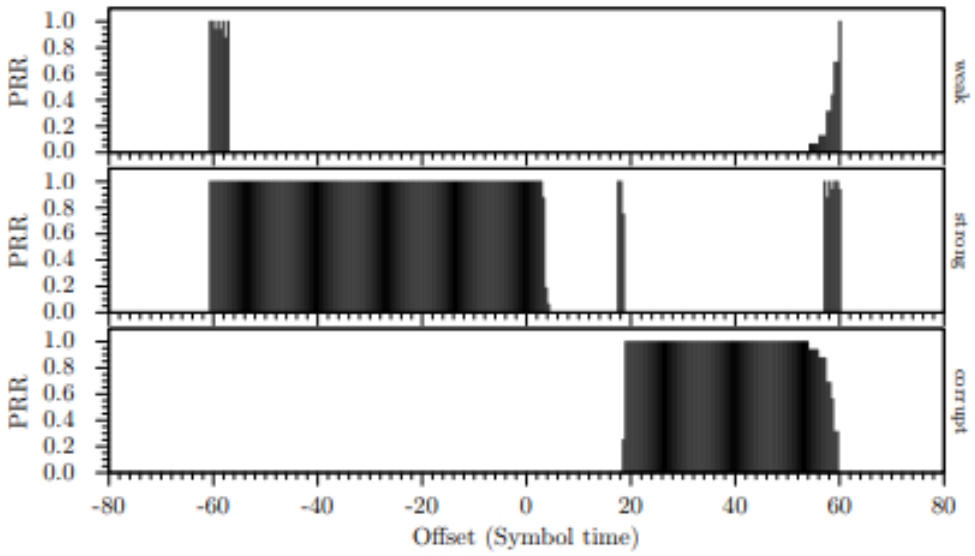

Figure 3.2: Example collision result. Spreading factor 11, bandwidth 125 kHz. X-axis shows the transmission offset relative to the weak node in symbol time, Y-axis shows the packet reception rate.

The experiment results are shown in Figure 3.2 for SF12 and BW 125 kHz. The Y-axis represents the packet reception rate. The X-axis represents the transmission offset relative to the weak node in symbol time. The top bar shows packets received from the weak transmitter at the receiver; the middle bar shows packets received from the strong transmitter at the receiver. The bottom bar shows when packets were received from either transmitter but deemed corrupt (Cyclic Redundancy Check (CRC) failure). We did notice that about 1 in 6000 packets was corrupted, but did not fail the CRC. Often these packets had 1 bit corrupted.

Results for other SF and BW combinations are very similar. Also, transmitting the same packet payload or a different payload does not change the obtained results significantly.
As can be seen, the strong transmitter is successfully decoded if it transmits not later than 3 symbol periods after the weak transmitter started . If the weak transmitter starts later than 3 symbol periods no transmission is received (or corrupted data is received).
Although the packet takes 60.25 symbol periods to transmit, both nodes can be received at an offset of −57 symbol periods or more. The tail of the strong node does destroy the initial preamble of the weak node, but as long as at most 3 symbols are destroyed, the weak packet can also be successfully received. This relationship is not symmetrical, as at an offset of +57 symbol periods, the weak node's tail (CRC) gets destroyed, invalidating a packet which may have been correctly received. It is only that at an offset of +60 symbol periods or more, both packets gets received perfectly.
We also experimented with two transmitters set to the same transmit power. In this case either of the two is perceived as stronger and the above described behaviour applies (although the role of stronger/weaker transmitter may alternate with each transmission making it difficult to conduct experiments and describe results).

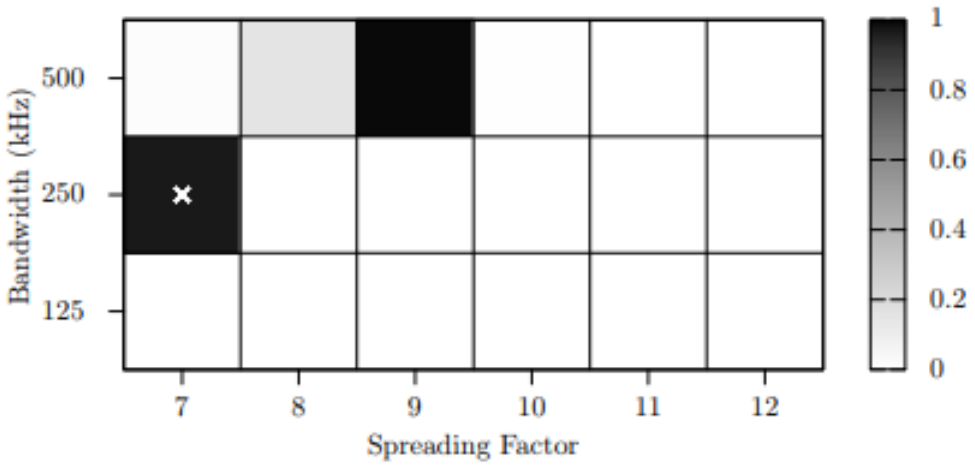

Figure 3.3: Carrier detection ratios for a transmitter sending at spreading factor 7 and bandwidth 250 kHz, indicated by the white cross. Carriers were also detected by adjacent data rates.

Findings
One of two concurrent transmission can be received with very high prob-

ability if both transmissions do not have an offset of more than 3 symbol periods. This translates to a duration between 768 µs and 98.3 ms, depending on the SF and BW. Synchronisation of nodes within these bounds is relatively easy to achieve and therefore protocols making use of this feature can easily be implemented with LoRa.

Carrier Activity Detection

Transceivers normally provide a CCA interface to detect an occupied channel. The CCA is used in communication protocols to decide if packets can be transmitted and to decide if the radio must be kept active to receive a message. In particular for power constrained nodes it is important to have an accurate and fast CCA mechanism as this enables implementation of power-efficient duty cycling. Nodes perform periodic short CCA checks and only power the receiver for longer if a transmission is detected.

LoRa transceivers do not provide a classical CCA interface based on an Received Signal Strength Indicator (RSSI) threshold to detect an occupied channel. LoRa can receive transmissions with a signal strength that is below the noise floor and, consequently, an RSSI threshold check will not reveal an occupied channel. LoRa radios provide therefore a CAD mode to detect a present preamble.
The CAD process takes approximately 2 symbol periods, and only requires the radio on for about 1 symbol period (The exact CAD duration is calculated as sum of (32/BW+(2^SF)/BW) seconds in RX mode and (SF $\cdot$ 2^SF)/(1750×103) seconds of processing). The processing phase requires about half of the energy required in receive mode (around 6 mA depending on the SF/BW). After receiving a 'CAD detected', a transceiver can be switched in RX mode to receive the ongoing transmission if required.

We set up an experiment to test the reliability of CAD. A detector node starts the CAD process on a regular interval (every 100 ms) and records whether it has detected a carrier or not. After 300 samples, it switches the SF/BW combination and repeats the process. A transmitter node is programmed to continuously send out preambles at a fixed SF/BW combination. The experiment is repeated with different transmitter SF/BW combinations.
The results for a transmitter using SF7 and a BW of 250 kHz are shown in Figure 3.3. Results for a transmitter using different SF/BW combinations are similar. When transmitter and receiver use the same SF/BW combi-

nation the worst detection rate was measured at 97% (for SF7 and a BW of 250 kHz). However, the CAD process also detects carriers in SF/BW combinations different from the combination the transmitter is using (up to 99% detections for SF9 and a BW of 500 kHz). This happens for data rates that are adjacent to the current data rate. When no transmitter is active, the detector had a false positive rate of 0.092%.

Findings
CAD can only detect channel occupancy while a preamble is transmitted. The detection probability is high (above 97%) and false positives are low (0.092%). However, if multiple LoRa networks are active on different SF/BW combinations false positives can be very high (depending on SF/BW ratios). When using multiple SF/BW combinations in the same network (or when constructing multiple networks separated by SF/BW) the choice of combinations is important when using CAD.

3.4 Applications of LORA

LoRa (Long Range) is a wireless communication technology that is widely used in Internet of Things (IoT) applications due to its long-range capabilities, low power consumption, and low data rates. This technology leverages the Chirp Spread Spectrum (CSS) modulation to achieve robust communication over several kilometers, making it ideal for a variety of applications across different industries. This report explores some of the key applications of LoRa technology.

Smart Cities

Smart Metering: LoRa is extensively used in smart metering for utilities such as electricity, water, and gas. Smart meters equipped with LoRa modules can transmit consumption data to utility companies at regular intervals. This enables more efficient energy management, real-time monitoring, and billing accuracy.

Environmental Monitoring: LoRa sensors can monitor environmental parameters such as air quality, temperature, humidity, and noise levels. This data can help city authorities manage pollution, plan green spaces, and enhance the overall quality of life for residents.

Smart Lighting: LoRa-based smart lighting systems allow for remote control and management of streetlights. These systems can adjust lighting based on time of day or activity levels, leading to energy savings and reduced operational costs.

Agriculture

Precision Farming: LoRa technology is pivotal in precision farming, where sensors are deployed to monitor soil moisture, temperature, and nutrient levels. This data helps farmers optimize irrigation, fertilization, and crop management, leading to increased yields and resource efficiency.

Livestock Tracking: LoRa-enabled devices can track the location and health of livestock. These devices can provide real-time data on the animals' movements, detect abnormalities in behavior, and help in managing herds more effectively.

Remote Monitoring: In remote agricultural areas, LoRa can be used to monitor weather conditions, water levels in irrigation systems, and the status of equipment. This helps farmers make informed decisions and reduce manual monitoring efforts.

Industrial IoT

- **Asset Tracking:** LoRa is used in industrial settings to track the location and status of assets such as machinery, vehicles, and tools. This enhances asset utilization, reduces losses, and improves maintenance schedules.
- **Predictive Maintenance:** By integrating LoRa sensors with industrial equipment, companies can monitor the health and performance of machines in real-time. This data can predict potential failures and schedule maintenance before breakdowns occur, minimizing downtime and maintenance costs.
- **Remote Monitoring:** LoRa technology enables remote monitoring of industrial processes, allowing operators to oversee operations from a central location. This is particularly useful in hazardous or hard-to-reach environments.

Healthcare

- **Remote Patient Monitoring:** LoRa can be used to monitor patients' vital signs such as heart rate, blood pressure, and glucose levels. This data can be transmitted to healthcare providers in real-time, enabling timely interventions and reducing hospital visits.
- **Medical Asset Tracking:** Hospitals can use LoRa technology to track the location of critical medical equipment such as wheelchairs, infusion pumps, and defibrillators. This ensures that equipment is available when needed and reduces time spent searching for assets.
- **Elderly Care:** LoRa-based wearable devices can monitor the health and location of elderly individuals. These devices can send alerts in case of falls or other emergencies, providing peace of mind to caregivers and family members.

Environmental Monitoring

- **Wildlife Tracking:** LoRa technology is used in wildlife conservation efforts to track the movements and behaviors of animals in their natural habitats. This data helps researchers understand animal patterns and implement conservation strategies.
- **Disaster Management:** LoRa sensors can monitor environmental conditions in areas prone to natural disasters such as floods, landslides, and wildfires. Early warning systems can be set up to alert authorities and communities, allowing for timely evacuations and preparations.
- **Water Quality Monitoring:** LoRa-enabled sensors can be deployed in water bodies to monitor parameters such as pH, turbidity, and pollutant levels. This data is crucial for maintaining water quality and managing resources effectively.

Supply Chain Management

- **Inventory Management:** LoRa technology is used in warehouses and distribution centers to track inventory levels and movements.

This helps in optimizing stock levels, reducing inventory costs, and improving order fulfillment.

- **Cold Chain Monitoring:** In the transportation of perishable goods such as food and pharmaceuticals, LoRa sensors monitor temperature and humidity levels. This ensures that products are stored and transported under optimal conditions, maintaining quality and compliance.

- **Fleet Management:** LoRa is used to monitor the location, speed, and condition of vehicles in a fleet. This enhances route optimization, reduces fuel consumption, and improves overall fleet efficiency.

LoRa technology has emerged as a versatile and robust solution for a wide range of applications across various industries. Its ability to provide long-range communication with low power consumption makes it ideal for IoT applications that require reliable and efficient data transmission. As the technology continues to evolve and its ecosystem expands, we can expect to see even more innovative applications and widespread adoption of LoRa in the coming years.

3.5 Security of LoRa

LoRa (Long Range) technology is widely used in IoT (Internet of Things) applications due to its long-range communication and low power consumption. However, like any communication technology, security is a crucial aspect to ensure the integrity, confidentiality, and availability of data transmitted over LoRa networks. This report delves into the security features, challenges, and best practices for securing LoRa networks.

Security Features of LoRa

- **End-to-End Encryption:**
  - LoRaWAN employs AES[1] 128-bit encryption to secure data between devices and the network server.
  - Two main keys are used:
    - **Network Session Key (NwkSKey):** Ensures data integrity and authentication between the end device and the network server.
    - **Application Session Key (AppSKey):** Encrypts payload data from the end device to the application server, ensuring confidentiality.
- **Device Authentication:**
  - Each LoRaWAN device has a unique Device EUI and an AppKey.
  - During activation, devices perform mutual authentication with the network to ensure authorized communication.
- **Frame Counter:**
  - LoRaWAN uses a frame counter to prevent replay attacks.
  - The network server checks the counter to verify message uniqueness.

[1] chapter4

Security Challenges

- **Physical Device Security:**
  LoRa devices, often deployed in remote or accessible locations, are susceptible to physical tampering. Attackers with physical access to a device can extract cryptographic keys, modify firmware, or disrupt operation.

- **Key Management:**
  Managing and distributing cryptographic keys securely is a critical challenge. Compromise of the AppKey or NwkSKey can lead to unauthorized access to the network and data.

- **Jamming and Interference:**
  LoRa operates in unlicensed ISM (Industrial, Scientific, and Medical) bands, making it susceptible to jamming and interference. An attacker can use a jamming device to disrupt communication by overwhelming the network with noise.

- **Eavesdropping:**
  Although LoRaWAN encrypts payload data, metadata such as device addresses and frame counters are transmitted in plaintext. Attackers can eavesdrop on this metadata to gain insights into network activity and potentially perform traffic analysis.

- **Replay Attacks:**
  Despite the use of frame counters to prevent replay attacks, improper handling or synchronization issues can still make networks vulnerable to this type of attack.

- **Physical Device Security:** LoRa devices, often deployed in remote or accessible locations, are susceptible to physical tampering. Attackers with physical access to a device can extract cryptographic keys, modify firmware, or disrupt operation.

- **Key Management:** Managing and distributing cryptographic keys securely is a critical challenge. Compromise of the AppKey or NwkSKey can lead to unauthorized access to the network and data.

- **Jamming and Interference:** LoRa operates in unlicensed ISM (Industrial, Scientific, and Medical) bands, making it susceptible to jamming and interference. An attacker can use a jamming device to disrupt communication by overwhelming the network with noise.
- **Eavesdropping:** Although LoRaWAN encrypts payload data, metadata such as device addresses and frame counters are transmitted in plaintext. Attackers can eavesdrop on this metadata to gain insights into network activity and potentially perform traffic analysis.
- **Replay Attacks:** Despite the use of frame counters to prevent replay attacks, improper handling or synchronization issues can still make networks vulnerable to this type of attack.

Best Practices for Securing LoRa Networks

- **Secure Device Provisioning:**
  - Ensure secure provisioning of devices by protecting the AppKey during manufacturing and distribution.
  - Use secure channels to transmit keys to devices and network servers.
- **Regular Key Rotation:**
  - Implement regular key rotation to minimize the impact of a compromised key.
  - Use Over-the-Air Activation (OTAA) to re-authenticate devices and provide new session keys.
- **Physical Security Measures:**
  - Deploy physical security measures to protect LoRa devices from tampering.
  - Use tamper-evident enclosures and regularly inspect devices for signs of physical interference.
- **Network Monitoring:**
  - Implement robust network monitoring to detect and respond to abnormal activities.

  - Watch for unusual traffic patterns, repeated join requests, or unexpected changes in frame counters.

- **Redundancy and Resilience:**
  - Design the network with redundancy and resilience in mind.
  - Use multiple gateways to provide overlapping coverage and reduce the impact of jamming or interference on a single gateway.

- **Secure Firmware Updates:**
  - Ensure secure firmware updates using cryptographic signing and verification.
  - Regularly update device firmware to patch vulnerabilities and enhance security features.

- **Privacy Measures:**
  - Implement privacy measures to protect metadata.
  - Consider techniques such as pseudonymization or encryption of metadata to prevent traffic analysis.

The security of LoRa networks is critical for the reliability and trustworthiness of IoT applications. While LoRaWAN provides robust security features such as end-to-end encryption and mutual authentication, challenges such as physical security, key management, and jamming need to be addressed. By following best practices and implementing strong security measures, it is possible to build secure and resilient LoRa networks that protect data integrity, confidentiality, and availability. As the technology and threat landscape evolve, continuous improvements and vigilance in security practices will be essential to maintaining the security of LoRa-based systems.

3.6 Future Directions of LoRa

LoRa (Long Range) technology has established itself as a fundamental component of the Internet of Things (IoT) ecosystem due to its long-range communication capabilities and low power consumption. As the demand for IoT applications continues to grow, the future of LoRa looks promising with several key directions for development and innovation. This report explores the potential future directions of LoRa technology.

Enhanced Network Scalability and Capacity

Network Densification: As the number of IoT devices increases, enhancing network scalability and capacity will be critical. Techniques such as network densification, where additional gateways are deployed to increase coverage and capacity, will help manage the growing traffic and ensure reliable communication.
Advanced Network Management: Advanced network management techniques, including dynamic allocation of network resources and improved congestion control mechanisms, will be necessary to optimize network performance and minimize latency in dense IoT deployments.

Integration with 5G and Other Technologies

5G Collaboration: Integration of LoRa with 5G networks can leverage the strengths of both technologies. While 5G provides high data rates and low latency, LoRa offers extensive coverage and low power consumption. Hybrid networks can combine these advantages to support a wider range of applications, from high-bandwidth to low-power IoT devices.
Interoperability with Other LPWAN Technologies: Future developments may focus on enhancing interoperability between LoRa and other Low Power Wide Area Network (LPWAN) technologies, such as NB-IoT (Narrowband IoT) and Sigfox. This would enable seamless connectivity and better resource utilization across different networks.

Improved Security Measures

Enhanced Cryptographic Algorithms: As security threats evolve, there will be a need for stronger cryptographic algorithms and more robust security protocols to protect LoRa networks. Future developments might include the implementation of quantum-resistant cryptographic techniques to safeguard against emerging threats.
AI-Driven Security: Artificial intelligence (AI) and machine learning (ML) can play a significant role in enhancing security. AI-driven security systems can detect and respond to anomalies in real-time, providing proactive defense mechanisms against cyber-attacks.

Advanced Data Analytics and Edge Computing

Edge Computing Integration: Integrating edge computing with LoRa networks can reduce latency and improve data processing efficiency. By

processing data closer to the source, edge computing can enhance real-time decision-making and reduce the load on central servers.
Big Data Analytics: The vast amount of data generated by LoRa devices presents opportunities for advanced data analytics. Big data analytics can uncover valuable insights, optimize operations, and enable predictive maintenance across various applications.

New and Expanded Applications

Smart Agriculture: Advancements in sensor technology and data analytics will expand the use of LoRa in precision agriculture. Future applications might include more sophisticated crop monitoring systems, automated irrigation control, and advanced livestock management solutions.
Smart Cities: The concept of smart cities will continue to evolve with LoRa technology at its core. Future applications could include integrated traffic management systems, enhanced public safety solutions, and more efficient energy management systems.
Healthcare Innovations: LoRa technology will enable new healthcare applications, such as advanced remote patient monitoring, real-time health analytics, and smart medical devices. These innovations can improve patient outcomes and reduce healthcare costs.

Environmental Monitoring and Conservation

Climate Change Monitoring: Future developments may see LoRa technology being used extensively for climate change monitoring. Advanced sensor networks can provide real-time data on environmental parameters, helping scientists and policymakers make informed decisions.
Wildlife Conservation: LoRa technology will continue to play a crucial role in wildlife conservation efforts. Future applications might include more sophisticated tracking systems for endangered species and better tools for monitoring habitats and ecosystems.

Enhanced Battery Life and Energy Harvesting

Ultra-Low Power Designs: Future LoRa devices will likely feature ultra-low power designs to extend battery life even further. Innovations in chip design and power management will enable devices to operate for longer periods on a single battery charge.
Energy Harvesting: Integrating energy harvesting technologies, such as

solar or kinetic energy, with LoRa devices can provide a sustainable power source, reducing the reliance on traditional batteries and minimizing maintenance.

The future of LoRa technology is bright, with numerous opportunities for growth and innovation. Enhancing network scalability, integrating with other technologies, improving security measures, and expanding applications across various sectors will drive the evolution of LoRa. As the IoT landscape continues to evolve, LoRa will remain a critical enabler of long-range, low-power communication, supporting a wide array of applications and contributing to the advancement of smart, connected solutions.

Final Thoughts

LoRa technology has emerged as a versatile and robust solution for a wide array of IoT applications. Its ability to provide long-range communication with low power consumption makes it ideal for numerous sectors, from smart cities and agriculture to industrial IoT and healthcare. As the technology evolves, continuous improvements in scalability, security, integration with other technologies, and application expansion will further cement LoRa's role in the IoT landscape. By addressing current challenges and leveraging future opportunities, LoRa will continue to enable innovative and efficient solutions, driving the advancement of smart, connected environments worldwide.

Chapter 4

AES

4.1 Motivation

In our project, the primary goal is to monitor the performance and behavior of large institutions to gather sufficient information for future improvements. This monitoring is crucial for optimizing operations, enhancing efficiency, and identifying potential areas of improvement within these institutions.

To achieve this, we have utilized LoRa Technology to establish connections between microcontrollers. LoRa (Long Range) technology is particularly well-suited for this application due to its capability to transmit data over long distances with low power consumption, making it ideal for creating a wide-area monitoring network without the need for extensive infrastructure.

As we progressed, we discovered that LoRa allows us to create an air-gapped system at certain stages. An air-gapped system is isolated from the internet and other external networks, which inherently adds an extra layer of security. This isolation is significant because it reduces the likelihood of external cyber threats, such as hacking attempts over the internet. By keeping the monitoring system within the range of the LoRa chip and disconnected from broader networks, we enhance the security and integrity of the data being transmitted.

However, this air-gapped system is not completely immune to potential security threats. Specifically, we must consider the possibility of an attacker being physically present within the institution and within the range of the LoRa technology. The frequency band used by LoRa SX1278 is publicly known, which means that the traffic between the transmitter and the receiver can be easily captured by anyone with the right equipment. If an attacker can capture this traffic, they can monitor and potentially control the entire system, leading to unauthorized access, data breaches, and

system manipulation.

Given this potential vulnerability, it is essential to secure the connection between the transmitter and receiver to protect the integrity and confidentiality of the data being transmitted. Without adequate security measures, the benefits of using an air-gapped LoRa system can be undermined by the threat of local interception and manipulation.

Therefore, our project includes a critical focus on implementing robust security mechanisms to safeguard the communication between LoRa modules. By integrating Advanced Encryption Standard (AES) encryption, we aim to ensure that even if the data is intercepted, it remains incomprehensible and useless to unauthorized parties. This encryption will not only protect sensitive information but also maintain the overall security and reliability of the monitoring system.

In summary, the motivation behind our project extends beyond just monitoring institutional performance; it encompasses the development of a secure, efficient, and resilient system that leverages LoRa technology's strengths while addressing its potential vulnerabilities through advanced security measures.

4.2 So, What is AES..?

Advanced Encryption Standard (AES) is a symmetric[1] encryption algorithm widely adopted for securing sensitive data. It is a block cipher that operates on fixed-size blocks of data, encrypting and decrypting information using a specified cryptographic key. AES is recognized for its efficiency, security, and versatility, making it a cornerstone in modern cryptography for applications ranging from secure communication and data storage to financial transactions and sensitive information protection. The algorithm supports key sizes of 128, 192, or 256 bits, with higher bit lengths offering increased security against brute-force attacks. AES has been endorsed by the U.S. National Institute of Standards and Technology (NIST) and is used globally as a standard encryption method in various industries and applications.

[1] Same key is used for encryption and decryption

AES Steps

1. **Plaintext Representation:** The plaintext is represented as a 16-byte array. For example, consider the plaintext "0123456789abcdef0123456789abcde where each character represents a hexadecimal digit (4 bits).

2. **Plaintext to Byte Array:** Convert the plaintext into a byte array:

$$\text{Plaintext} = \begin{bmatrix} 01 & 23 & 45 & 67 & 89 & ab & cd & ef & 01 & 23 \\ 45 & 67 & 89 & ab & cd & ef & & & & \end{bmatrix}$$

3. **Byte Array to State Matrix:** Map the byte array into a 4x4 matrix. The mapping is done column-wise:

$$\text{State} = \begin{bmatrix} \text{byte}_0 & \text{byte}_4 & \text{byte}_8 & \text{byte}_{12} \\ \text{byte}_1 & \text{byte}_5 & \text{byte}_9 & \text{byte}_{13} \\ \text{byte}_2 & \text{byte}_6 & \text{byte}_{10} & \text{byte}_{14} \\ \text{byte}_3 & \text{byte}_7 & \text{byte}_{11} & \text{byte}_{15} \end{bmatrix}$$

   For our example:

$$\text{State} = \begin{bmatrix} 01 & 89 & 01 & 89 \\ 23 & ab & 23 & ab \\ 45 & cd & 45 & cd \\ 67 & ef & 67 & ef \end{bmatrix}$$

Steps Explained

KeyExpansion

The key expansion step generates a series of round keys from the cipher key K. These keys are used in the AddRoundKey step of each round.

AddRoundKey

This step involves a bitwise XOR operation between the state and a round key derived from the cipher key.

SubBytes

In this step, each byte in the state is replaced with its corresponding value from a fixed 8-bit lookup table known as the S-box.

Algorithm 2 AES Algorithm Steps

Require: Plaintext P, Cipher Key K
Ensure: Ciphertext C

```
1: procedure AES(P, K)
2:     W ← KeyExpansion(K)                          ▷ Expand the key
3:     S ← P ⊕ W[0]                                 ▷ Initial AddRoundKey step
4:     for i = 1 to Nr − 1 do                       ▷ Nr = Number of rounds
5:         S ← SubBytes(S)
6:         S ← ShiftRows(S)
7:         S ← MixColumns(S)
8:         S ← S ⊕ W[i]                             ▷ AddRoundKey step
9:     end for
10:    S ← SubBytes(S)
11:    S ← ShiftRows(S)
12:    S ← S ⊕ W[Nr]                                ▷ Final AddRoundKey step
13:    return S
14: end procedure
```

ShiftRows

This step cyclically shifts the bytes in each row of the state. The number of shifts depends on the row index.

MixColumns

In this step, the columns of the state are treated as polynomials over $\mathbb{GF}(2^8)$ and are multiplied by a fixed polynomial.

Number of Rounds

The number of rounds Nr depends on the key size:

- 10 rounds for 128-bit keys.
- 12 rounds for 192-bit keys.
- 14 rounds for 256-bit keys.

Numerical Example

Consider the following example with a 128-bit key and plaintext:

- Plaintext P (in hexadecimal): `32 88 31 e0 43 5a 31 37 f6 30 98 07 a8 8d a2 34`
- Cipher Key K (in hexadecimal): `2b 7e 15 16 28 ae d2 a6 ab f7 33 66 17 88 09 cf`

Initial AddRoundKey Step

Convert plaintext and key to state matrix:

$$P = \begin{bmatrix} 32 & 88 & 31 & e0 \\ 43 & 5a & 31 & 37 \\ f6 & 30 & 98 & 07 \\ a8 & 8d & a2 & 34 \end{bmatrix}$$

$$K = \begin{bmatrix} 2b & 7e & 15 & 16 \\ 28 & ae & d2 & a6 \\ ab & f7 & 33 & 66 \\ 17 & 88 & 09 & cf \end{bmatrix}$$

Perform XOR operation:

$$S = P \oplus K = \begin{bmatrix} 32 & 88 & 31 & e0 \\ 43 & 5a & 31 & 37 \\ f6 & 30 & 98 & 07 \\ a8 & 8d & a2 & 34 \end{bmatrix} \oplus \begin{bmatrix} 2b & 7e & 15 & 16 \\ 28 & ae & d2 & a6 \\ ab & f7 & 33 & 66 \\ 17 & 88 & 09 & cf \end{bmatrix} = \begin{bmatrix} 19 & f6 & 24 & f6 \\ 6b & 5a & b5 & 91 \\ 5d & 97 & cb & 61 \\ bf & 33 & 4f & fb \end{bmatrix}$$

SubBytes Step

Apply the S-box substitution to each byte in the state matrix.

$$S' = \begin{bmatrix} d4 & 27 & 11 & ae \\ e0 & bf & 98 & f1 \\ b8 & b4 & 5d & e5 \\ 1e & 41 & 52 & 30 \end{bmatrix}$$

ShiftRows Step

Shift the rows of the state matrix.

$$S'' = \begin{bmatrix} d4 & 27 & 11 & ae \\ bf & 98 & f1 & e0 \\ 5d & e5 & b8 & b4 \\ 30 & 1e & 41 & 52 \end{bmatrix}$$

MixColumns Step

Mix the columns of the state matrix.

$$S''' = \begin{bmatrix} 04 & 66 & 81 & e5 \\ e0 & cb & 19 & 9a \\ 48 & f8 & d3 & 7a \\ 28 & 06 & 26 & 4c \end{bmatrix}$$

AddRoundKey Step

Apply the next round key (example only includes one key for brevity).

$$W[1] = \begin{bmatrix} a0 & 88 & 23 & 2a \\ fa & 54 & a3 & 6c \\ fe & 2c & 39 & 76 \\ 17 & b1 & 39 & 05 \end{bmatrix}$$

$$S'''' = S''' \oplus W[1] = \begin{bmatrix} 04 & 66 & 81 & e5 \\ e0 & cb & 19 & 9a \\ 48 & f8 & d3 & 7a \\ 28 & 06 & 26 & 4c \end{bmatrix} \oplus \begin{bmatrix} a0 & 88 & 23 & 2a \\ fa & 54 & a3 & 6c \\ fe & 2c & 39 & 76 \\ 17 & b1 & 39 & 05 \end{bmatrix} = \begin{bmatrix} a4 & ee & a2 & cf \\ 1a & 9f & ba & f6 \\ b6 & d4 & ea & 0c \\ 3f & b7 & 1f & 49 \end{bmatrix}$$

This process repeats for the remaining rounds until the final ciphertext is obtained.

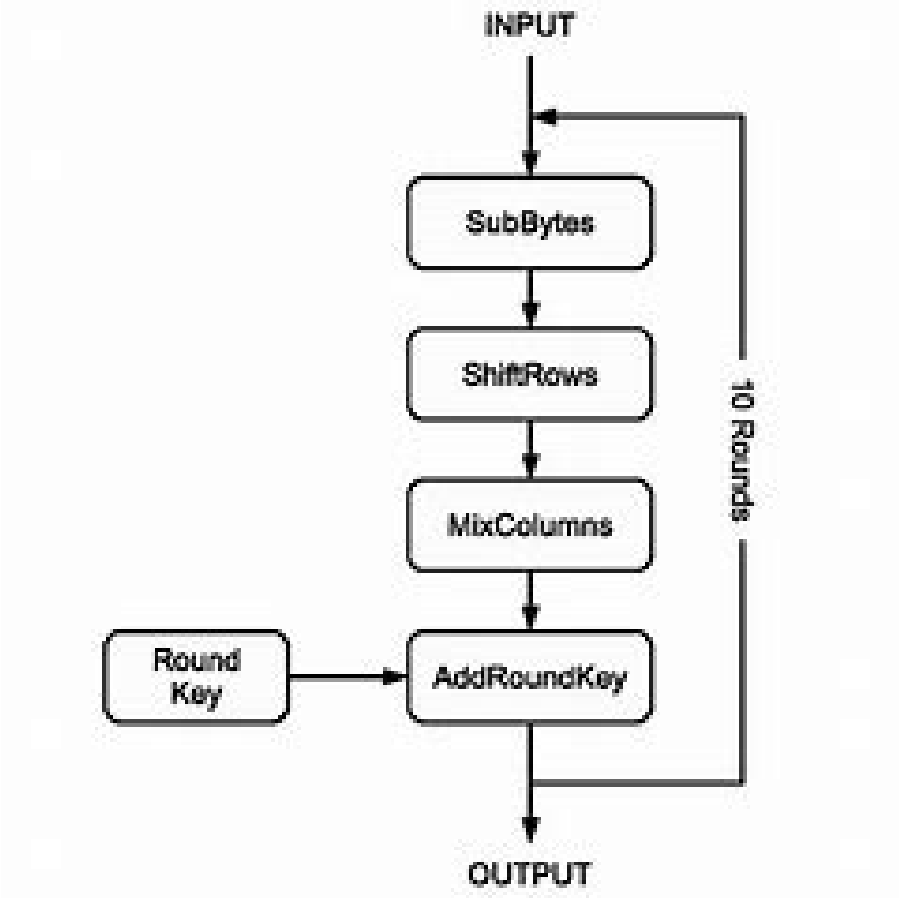

Figure 4.1: Simple block diagram

Simplified C++ Code for AES Algorithm

```
// S-box transformation table
const unsigned char sbox[256] = {
    // 0 1 2 3 4 5 6 7 8 9 A B C D E F
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0
       x2b, 0xfe, 0xd7, 0xab, 0x76,
    // ... (rest of the S-box values)
};

// Rcon table
const unsigned char Rcon[255] = {
    0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36,
    // ... (rest of the Rcon values)
};

// Key expansion function
vector<unsigned char> KeyExpansion(const vector<unsigned char>& key) {
    vector<unsigned char> expandedKey(Nb * (Nr + 1) * 4);
```

```
    for (int i = 0; i < Nk * 4; ++i) {
        expandedKey[i] = key[i];
    }

    for (int i = Nk; i < Nb * (Nr + 1); ++i) {
        vector<unsigned char> temp(4);
        for (int j = 0; j < 4; ++j) {
            temp[j] = expandedKey[(i - 1) * 4 + j];
        }
        if (i % Nk == 0) {
            unsigned char t = temp[0];
            temp[0] = sbox[temp[1]] ^ Rcon[i / Nk];
            temp[1] = sbox[temp[2]];
            temp[2] = sbox[temp[3]];
            temp[3] = sbox[t];
        }
        for (int j = 0; j < 4; ++j) {
            expandedKey[i * 4 + j] = expandedKey[(i - Nk) * 4 + j] ^ temp[
                j];
        }
    }

    return expandedKey;
}

// AddRoundKey step
void AddRoundKey(State& state, const vector<unsigned char>& roundKey, int
     round) {
    for (int r = 0; r < 4; ++r) {
        for (int c = 0; c < 4; ++c) {
            state[r][c] ^= roundKey[(round * Nb * 4) + (c * Nb) + r];
        }
    }
}

// SubBytes step
void SubBytes(State& state) {
    for (int r = 0; r < 4; ++r) {
        for (int c = 0; c < 4; ++c) {
            state[r][c] = sbox[state[r][c]];
        }
    }
}

// ShiftRows step
void ShiftRows(State& state) {
    unsigned char temp;
    temp = state[1][0];
```

```
    state[1][0] = state[1][1];
    state[1][1] = state[1][2];
    state[1][2] = state[1][3];
    state[1][3] = temp;

    temp = state[2][0];
    state[2][0] = state[2][2];
    state[2][2] = temp;
    temp = state[2][1];
    state[2][1] = state[2][3];
    state[2][3] = temp;

    temp = state[3][0];
    state[3][0] = state[3][3];
    state[3][3] = state[3][2];
    state[3][2] = state[3][1];
    state[3][1] = temp;
}

// MixColumns step
void MixColumns(State& state) {
    // ... (implementation of MixColumns)
}

// Encrypt function
void Encrypt(State& state, const vector<unsigned char>& key) {
    vector<unsigned char> expandedKey = KeyExpansion(key);

    AddRoundKey(state, expandedKey, 0);

    for (int round = 1; round < Nr; ++round) {
        SubBytes(state);
        ShiftRows(state);
        MixColumns(state);
        AddRoundKey(state, expandedKey, round);
    }

    SubBytes(state);
    ShiftRows(state);
    AddRoundKey(state, expandedKey, Nr);
}

int main() {
    State state = {
        {0x32, 0x88, 0x31, 0xe0},
        {0x43, 0x5a, 0x31, 0x37},
        {0xf6, 0x30, 0x98, 0x07},
        {0xa8, 0x8d, 0xa2, 0x34}
```

```
    };

    vector<unsigned char> key = {
        0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7, 0x1a,
            0x88, 0x09, 0xcf, 0x4f, 0x3c
    };

    Encrypt(state, key);

    cout << "Encrypted␣State:" << endl;
    for (int r = 0; r < 4; ++r) {
        for (int c = 0; c < 4; ++c) {
            printf("%02x␣", state[r][c]);
        }
        cout << endl;
    }

    return 0;
}
```

Encryption

```
#include <SPI.h>
#include <LoRa.h>
#include <DHT.h>
#include <MQUnifiedsensor.h>
#include <AESLib.h>

#define DHTPIN 3
#define DHTTYPE DHT22
#define Board "Arduino UNO"
#define MQ_PIN A0
#define Voltage_Resolution 5
#define ADC_Bit_Resolution 10

DHT dht(DHTPIN, DHTTYPE);
MQUnifiedsensor MQ4(Board, Voltage_Resolution, ADC_Bit_Resolution, MQ_PIN
    , "MQ-4");
AESLib aesLib;

byte aes_key[] = { 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB,
    0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C };
byte aes_iv[] = { 0x3D, 0xAF, 0xBA, 0x42, 0x9E, 0x6B, 0x5F, 0x70, 0x1A, 0
    xE3, 0x9A, 0x2C, 0x76, 0x2D, 0x1A, 0x6E };

void setup() {
```

```
  Serial.begin(9600);
  while (!Serial);

  Serial.println("LoRa␣Sender␣with␣DHT22␣and␣MQ4␣Sensors");

  if (!LoRa.begin(434E6)) {
    Serial.println("Starting␣LoRa␣failed!");
    while (1);
  }

  dht.begin();
  MQ4.setRegressionMethod(1);
  MQ4.init();

  float calcR0 = 0;
  for (int i = 1; i <= 10; i++) {
    MQ4.update();
    calcR0 += MQ4.calibrate(4.4); // RatioMQ4CleanAir = 4.4 (for clean
       air)
    delay(500);
  }
  MQ4.setR0(calcR0 / 10);
  Serial.println("MQ4␣Calibration␣done.");
}

void loop() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  MQ4.update();
  float LPG = MQ4.readSensor();
  float CH4 = MQ4.readSensor();
  float CO = MQ4.readSensor();
  float Alcohol = MQ4.readSensor();
  float Smoke = MQ4.readSensor();

  String message = "Temperature:␣" + String(temperature) + "°C,␣Humidity:
     ␣" + String(humidity) + "%␣|␣Gas␣Readings:␣";
  message += "LPG=" + String(LPG) + "␣ppm,␣CH4=" + String(CH4) + "␣ppm,␣
     CO=" + String(CO) + "␣ppm,␣Alcohol=" + String(Alcohol) + "␣ppm,␣
     Smoke=" + String(Smoke) + "␣ppm";

  String encryptedMessage = encryptMessage(message);

  Serial.print("Encrypted␣message:␣");
  Serial.println(encryptedMessage);

  LoRa.beginPacket();
```

```
  LoRa.print(encryptedMessage);
  LoRa.endPacket();

  delay(5000);
}

String encryptMessage(String message) {
  int messageLength = message.length();
  byte plaintext[messageLength + 1];
  message.getBytes(plaintext, messageLength + 1);

  byte ciphertext[messageLength + 1];
  uint16_t len = aesLib.encrypt(plaintext, messageLength, ciphertext,
     aes_key, 128, aes_iv);

  String encryptedHex = "";
  for (int i = 0; i < len; i++) {
    encryptedHex += String(ciphertext[i], HEX);
  }

  return encryptedHex;
}
```

Decryption

```
#include <SPI.h>
#include <LoRa.h>
#include <AESLib.h>

AESLib aesLib;

byte aes_key[] = { 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB,
    0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C };
byte aes_iv[] = { 0x3D, 0xAF, 0xBA, 0x42, 0x9E, 0x6B, 0x5F, 0x70, 0x1A, 0
    xE3, 0x9A, 0x2C, 0x76, 0x2D, 0x1A, 0x6E };

void setup() {
  Serial.begin(9600);
  while (!Serial);

  Serial.println("LoRa␣Receiver␣with␣AES␣Decryption");
  if (!LoRa.begin(434E6)) {
    Serial.println("Starting␣LoRa␣failed!");
    while (1);
  }
}
```

```
void loop() {
  // Try to parse packet
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    // Received a packet
    Serial.print("Received␣packet:␣");

    // Read packet into a String
    String received = "";
    while (LoRa.available()) {
      received += (char)LoRa.read();
    }

    Serial.println(received);

    // Decrypt the message
    String decryptedMessage = decryptMessage(received);
    Serial.print("Decrypted␣message:␣");
    Serial.println(decryptedMessage);

    // Wait before next packet (if needed)
    delay(1000);
  }
}

String decryptMessage(String encryptedHex) {
  int len = encryptedHex.length() / 2;
  byte ciphertext[len];
  for (int i = 0; i < len; i++) {
    ciphertext[i] = strtoul(encryptedHex.substring(i * 2, i * 2 + 2).
       c_str(), NULL, 16);
  }

  byte plaintext[len];
  uint16_t decryptedLength = aesLib.decrypt(ciphertext, len, plaintext,
     aes_key, 128, aes_iv);

  // Convert decrypted bytes to a String
  String decryptedMessage = "";
  for (int i = 0; i < decryptedLength; i++) {
    decryptedMessage += (char)plaintext[i];
  }

  return decryptedMessage;
}
```

Figure 4.2: Encrypted packets are sent

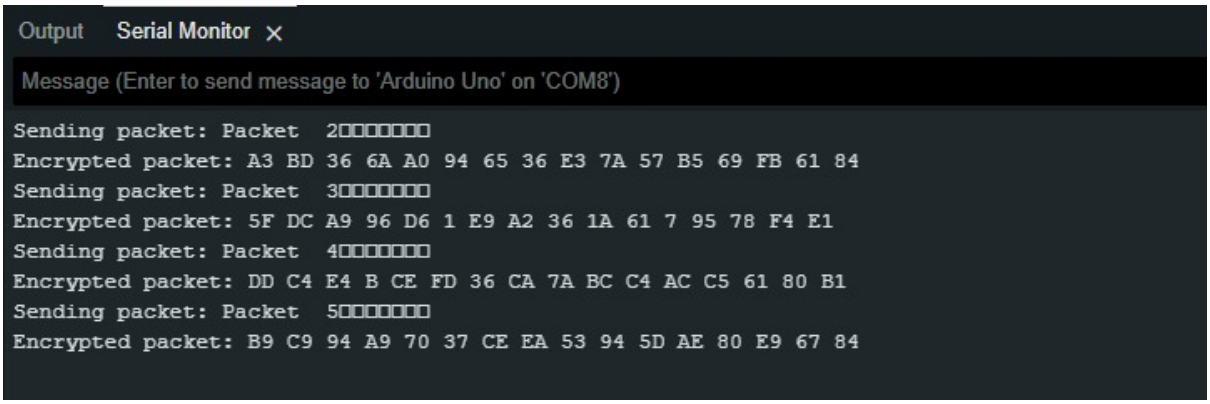

Figure 4.3: Decrypted Packets

Chapter 5

Application

5.1 LoRa (using SX1278) and YOLO on STM32 for Real-time Performance and Safety Monitoring

LoRa (using SX1278) Technology

The SX1278 is a popular LoRa transceiver module used for long-range communication. Here's how we implemented it in our system:

Sensor Integration

- **Sensors**: Various sensors (temperature, humidity, vibration, gas, etc.) are deployed throughout the factory.

- **LoRa Transceivers**: Each sensor node is equipped with an SX1278 LoRa transceiver module for wireless communication.

Data Transmission

- **Wireless Communication**: Sensor data is transmitted wirelessly using the SX1278 modules. These modules are configured to operate at a suitable frequency (e.g., 433 MHz) to ensure optimal performance in the factory environment.

- **Data Packets**: Data from sensors is packaged into small data packets and transmitted over the air to a central receiver.

Central Receiver

- **STM32 Microcontroller**: An STM32 microcontroller with an SX1278 module acts as the central receiver, collecting data from multiple sensor nodes.
- **Data Aggregation**: The STM32 processes and aggregates the data for real-time analysis.

Network Configuration

- **Point-to-Point Communication**: Each sensor node communicates directly with the central receiver, simplifying the network architecture and reducing latency.
- **Error Handling**: The system includes mechanisms for error detection and correction to ensure reliable data transmission.

YOLO (You Only Look Once) Algorithm

YOLO is an advanced, real-time object detection algorithm that enhances safety monitoring in our system. Here's how it is integrated with the STM32:

Object Detection

- **Camera Integration**: Cameras are strategically placed to cover critical areas in the factory.
- **Image Processing**: The camera feeds are processed using the YOLO algorithm to detect objects such as machinery, tools, and workers.

Algorithm Execution

- **Edge Computing**: The YOLO algorithm is implemented on the STM32 microcontroller. Due to the computational requirements of YOLO, a high-performance STM32 variant (e.g., STM32H7 series) is used.
- **Model Optimization**: The YOLO model is optimized for the STM32 platform to ensure it runs efficiently with limited resources. Techniques such as quantization and model pruning are applied.

Safety Monitoring

- **Hazard Detection**: YOLO identifies potential hazards in real-time (e.g., workers entering restricted areas, malfunctioning machinery).
- **Alerts**: When a potential hazard is detected, the system generates alerts and takes predefined actions (e.g., shutting down machinery, notifying supervisors).

Performance Monitoring

- **Activity Tracking**: YOLO tracks worker activities to ensure compliance with safety protocols and operational procedures.
- **Data Analysis**: The data collected from YOLO is analyzed to improve safety protocols and enhance operational efficiency.

System Integration

The integration of LoRa (using SX1278) and YOLO on STM32 creates a comprehensive real-time performance and safety monitoring system:

- **Data Fusion**: Sensor data transmitted via LoRa and visual data processed by YOLO are combined to provide a holistic view of the factory environment.
- **Real-time Processing**: The STM32 microcontroller handles real-time data processing and decision-making, ensuring quick responses to any safety or performance issues.
- **User Interface**: A user-friendly interface displays real-time data, alerts, and analytics, allowing factory managers to monitor and manage the environment effectively.

This setup ensures that our system can efficiently monitor factory performance and safety in real-time, leveraging the strengths of LoRa communication and the YOLO algorithm on the robust STM32 platform.

Here are the projected figures for our project's expected output.

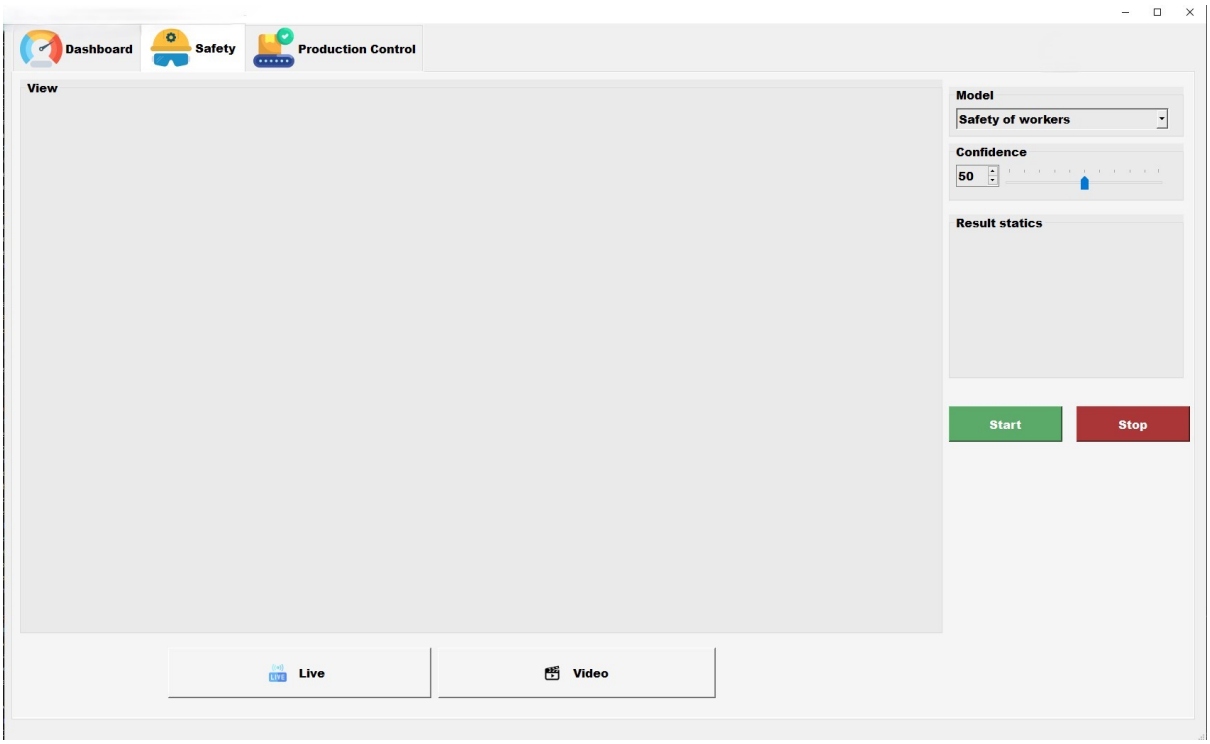

Figure 5.1: First page of GUI

In the figure above, three taps are utilized to monitor and display an event.

- **Dashboard** - displays readings from sensors.
- **Safety** - monitors the safety of workers in institutions.
- **Production control** - tracks item quantities and detects defects.

Application can process live stream or recorded video.

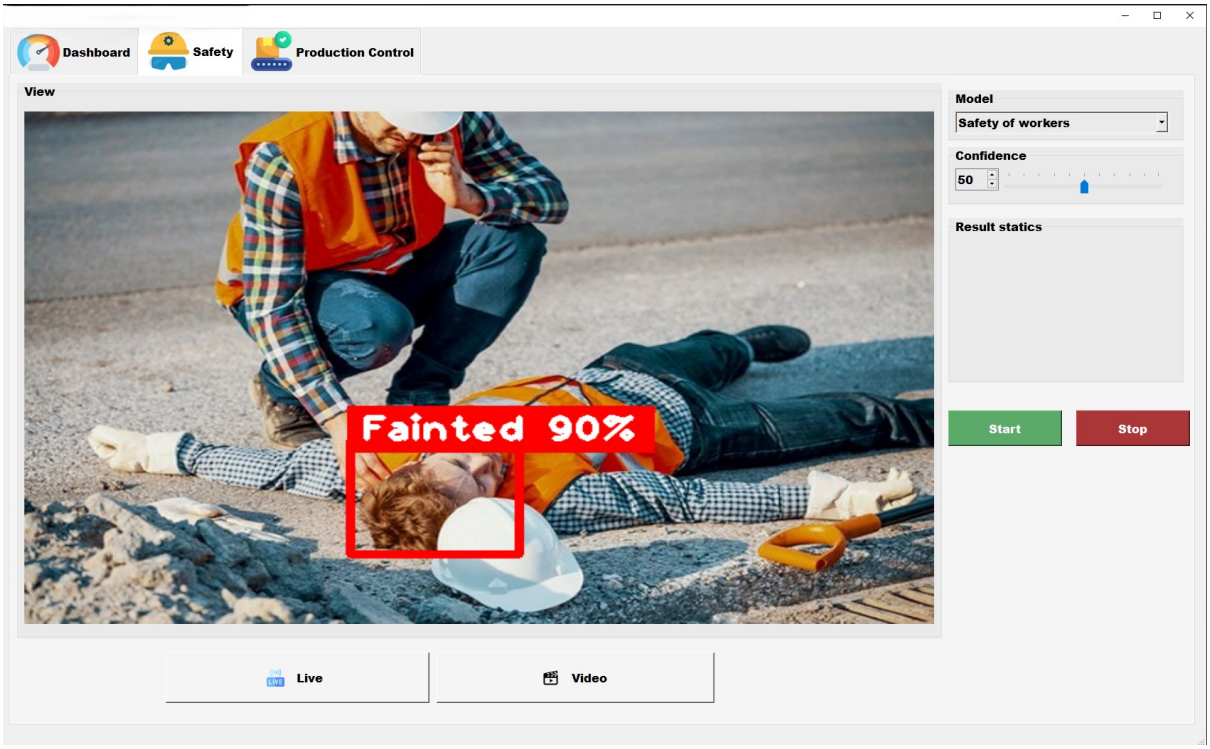

Figure 5.2: Application capture a fainted worker in the organisation

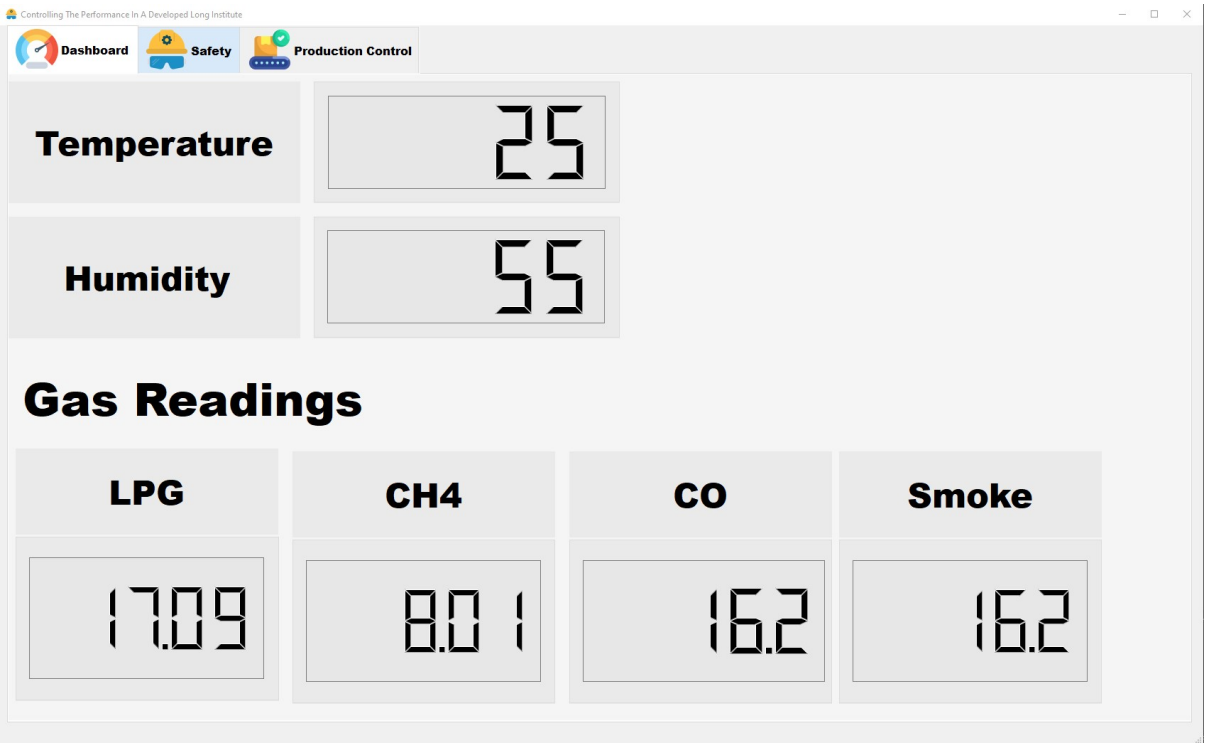

Figure 5.3: For the sake of safety, sensors readings has to monitored

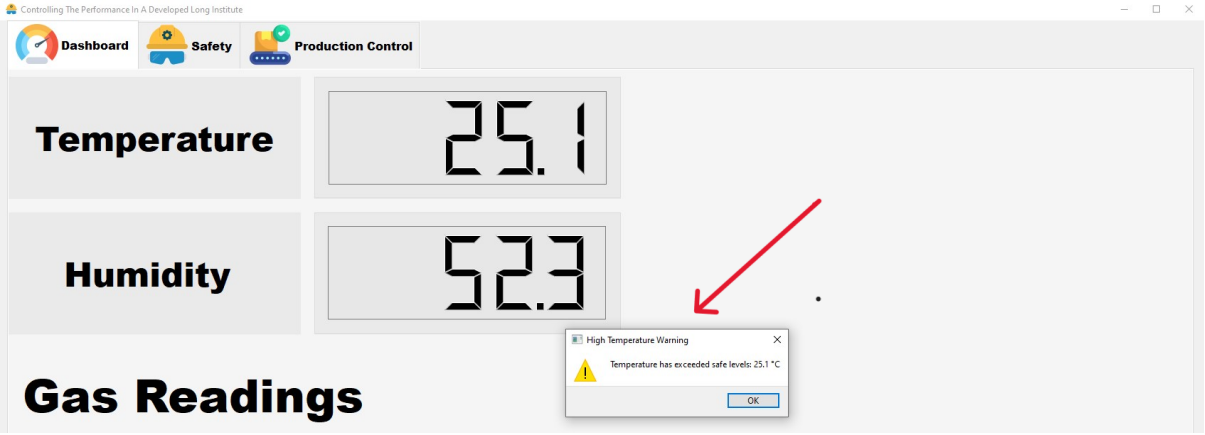

Figure 5.4: WARNING!!!!!!!!!!

WARNINGS!!!! This happens when the value of any reading exceeds a certain threshold.

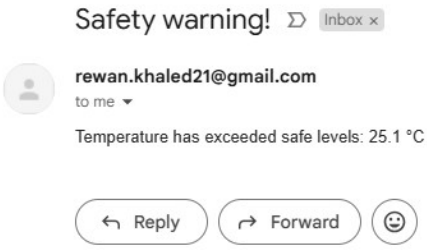

Figure 5.5: An email will be sent to the manufacturer to assist them in taking the necessary procedures following the previous warning.

So, this was the mechanism of how our application works

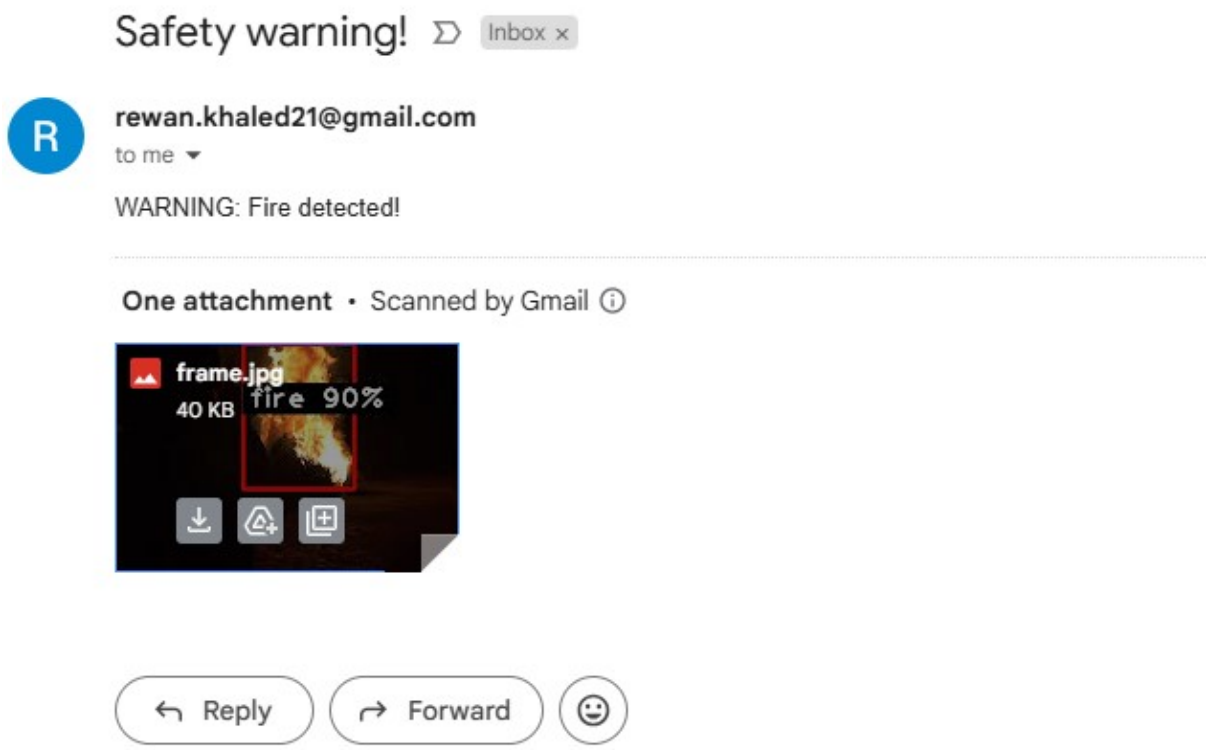

Figure 5.6: The email includes screen shots of whatever event triggered the Alarm

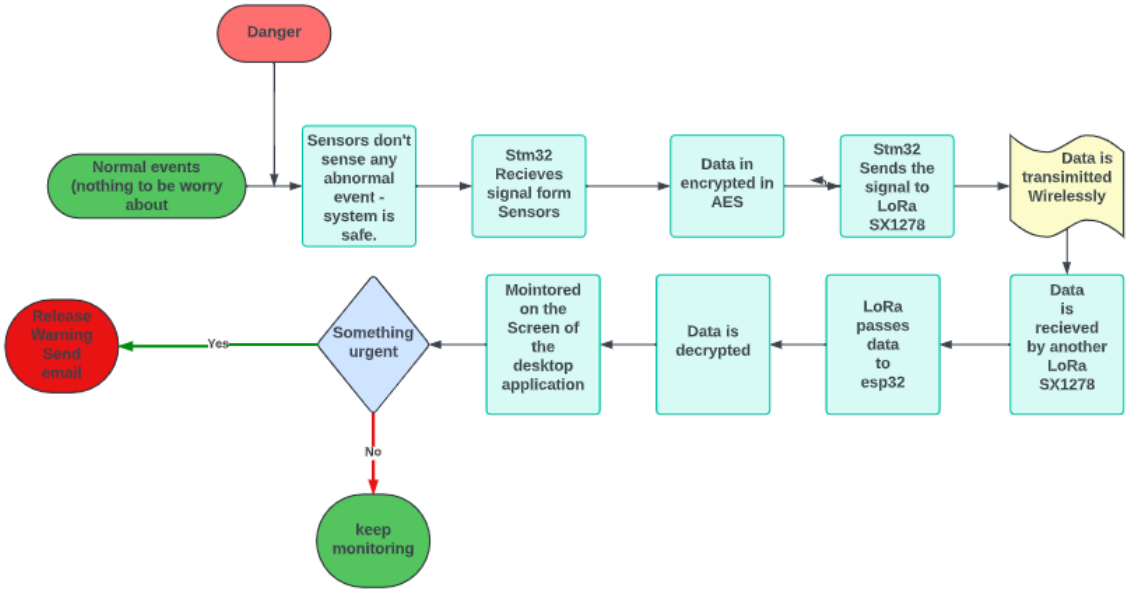

Figure 5.7: summary

Chapter 6

Future work

- **Worker Health Monitoring:**

Integrate wearable biometric sensors (e.g., heart rate monitors, accelerometers) with IoT devices to continuously monitor workers' vital signs. This data can be analyzed in real-time to detect anomalies like fainting or drowsiness more accurately. Machine learning algorithms could be employed to establish personalized baselines for each worker, enhancing the detection accuracy.

Health monitoring in industrial settings involves the systematic collection and analysis of physiological data from workers to ensure their safety and well-being. The principles underlying effective health monitoring encompass both technological and ethical considerations. Technologically, the integration of wearable sensors and IoT devices enables continuous monitoring of vital signs such as heart rate, body temperature, and activity levels. This real-time data acquisition facilitates early detection of anomalies like fatigue or stress, crucial for preemptive intervention to prevent accidents or health crises. Ethically, principles of consent, privacy, and data security are paramount. Workers must be informed and consenting participants in health monitoring programs, with clear policies in place to safeguard their personal information. Moreover, adherence to ethical guidelines ensures that monitoring prac-

tices prioritize worker autonomy and dignity, while promoting a supportive work environment conducive to both productivity and health. As advancements in sensor technology and data analytics continue to evolve, integrating these principles will be essential in fostering a harmonious balance between technological innovation and ethical responsibility in industrial health monitoring systems.

Integrate wearable biometric sensors (e.g., heart rate monitors, accelerometers) with IoT devices to continuously monitor workers' vital signs. This data can be analyzed in real-time to detect anomalies like fainting or drowsiness more accurately. Machine learning algorithms could be employed to establish personalized baselines for each worker, enhancing the detection accuracy. Health monitoring in industrial settings involves the systematic collection and analysis of physiological data from workers to ensure their safety and well-being. The principles underlying effective health monitoring encompass both technological and ethical considerations. Technologically, the integration of wearable sensors and IoT devices enables continuous monitoring of vital signs such as heart rate, body temperature, and activity levels. This real-time data acquisition facilitates early detection of anomalies like fatigue or stress, crucial for preemptive intervention to prevent accidents or health crises. Ethically, principles of consent, privacy, and data security are paramount. Workers must be informed and consenting participants in health monitoring programs, with clear policies in place to safeguard their personal information. Moreover, adherence to ethical guidelines ensures that monitoring practices prioritize worker autonomy and dignity, while promoting a supportive work environment conducive to both productivity and health. As advancements in sensor technology and data analytics continue to evolve, integrating these principles will be essential in fostering a harmonious balance between technological innovation

and ethical responsibility in industrial health monitoring systems.

We can use:

- **Heart rate sensor:**

Shown in figure 6.1

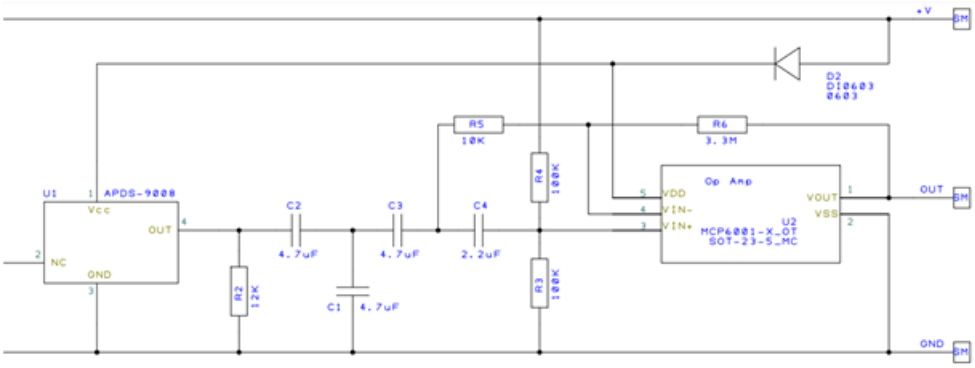

Figure 6.1: Heart rate sensor circuit diagram with signal conditioning and amplification using an operational amplifier.

- **Accelerometer sensor:**

Shown in figure 6.2

- **Predictive Maintenance:**

Utilize data from sensors (e.g., DHT11 for temperature and humidity, MQ4 for gas detection) to implement predictive maintenance algorithms. By analyzing sensor data over time, it becomes possible to predict equipment failures before they occur, thereby minimizing downtime and optimizing production efficiency.

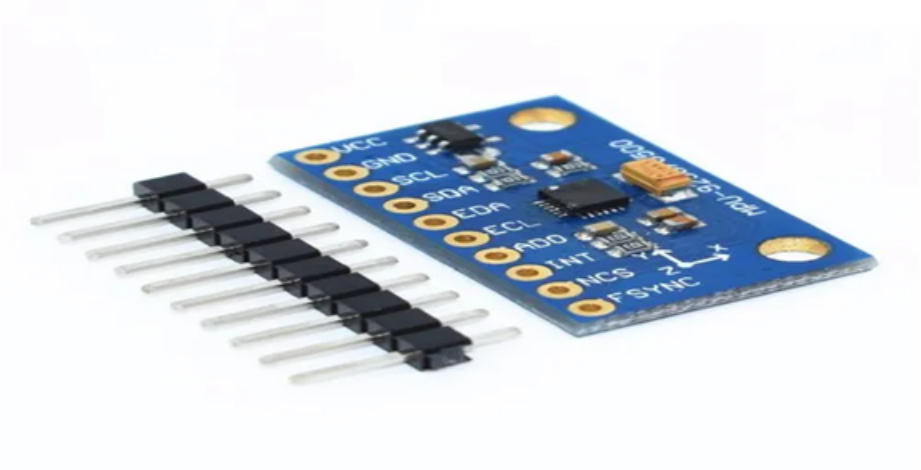

Figure 6.2: Accelerometer sensor module with breakout pins for easy interfacing and measurement of acceleration in three axes (X, Y, Z).

Predictive maintenance (PdM) represents a proactive approach to maintenance management that leverages data analytics and machine learning to predict equipment failures before they occur. The principles underlying effective predictive maintenance systems involve a combination of data acquisition, analysis, and action. Firstly, data acquisition entails the continuous monitoring of equipment parameters using sensors capable of measuring variables such as temperature, vibration, and fluid levels. These sensors generate vast amounts of data, which are then processed and analyzed using machine learning algorithms. These algorithms identify patterns and trends indicative of potential equipment degradation or failure. Secondly, effective predictive maintenance systems emphasize the importance of timely and accurate analysis. Advanced analytics techniques, including statistical modeling, anomaly detection, and pattern recognition, enable the identification of early warning signs of equipment deterioration. Thirdly, actionable insights derived from predictive maintenance analytics guide maintenance strategies, allowing for preemptive repairs or replacements during scheduled downtimes. Moreover, integrating predictive maintenance systems with existing enterprise

resource planning (ERP) systems facilitates seamless planning and resource allocation, optimizing operational efficiency. By adhering to these principles, organizations can minimize unplanned downtime, reduce maintenance costs, and extend equipment lifespan, thereby enhancing overall operational reliability and productivity in industrial settings.

- **Advantages of predictive maintenance**

Compared with preventive maintenance, predictive maintenance ensures that a piece of equipment requiring maintenance is only shut down right before imminent failure. This reduces the total time and cost spent maintaining equipment.

This brings several cost savings:
- Minimizing the time, the equipment is being maintained
- Minimizing the production hours lost to maintenance
- Minimizing the cost of spare parts and supplies

In addition to these advantages predictive maintenance also:
- Increase asset life: Regularly monitoring equipment health and addressing minor issues before they become major can extend the useful life of assets.
- Optimizes maintenance activities: Instead of routine or scheduled maintenance (which might be over or underdone), PdM ensures maintenance is carried out only when required, leading to efficient use of resources.
- Allows for better spare parts management: Knowing in advance what parts might fail allows for better inventory management, reducing the need for overstocking and ensuring parts are available when needed.

Who uses predictive maintenance?

Generally speaking, a maintenance manager and maintenance team use predictive maintenance tools and asset management systems to monitor impending equipment failure and maintenance tasks.

How is predictive maintenance used?

Let's say you have a pump on your production line. If this pump breaks, it will stall production until you can fix or replace it, which could take hours. Your asset management system can monitor the pump's temperature. If its temperature rises past a certain threshold, you know the pump is under stress and could possibly fail soon. You can then schedule some time to perform preventive maintenance before a complete failure stops production.

Predictive maintenance software can notify the maintenance team of the stress on a specific machine. It uses predictive analytics to flag issues and lets the team know to set up preventative maintenance, which helps reduce costly downtime.

Advantages of predictive maintenance

Compared with preventive maintenance, predictive maintenance ensures that a piece of equipment requiring maintenance is only shut down right before imminent failure. This reduces the total time and cost spent maintaining equipment.

This brings several cost savings:

- Minimizing the time, the equipment is being maintained

- Minimizing the production hours lost to maintenance
- Minimizing the cost of spare parts and supplies

In addition to these advantages predictive maintenance also:

- Increase asset life: Regularly monitoring equipment health and addressing minor issues before they become major can extend the useful life of assets.
- Optimizes maintenance activities: Instead of routine or scheduled maintenance (which might be over or underdone), PdM ensures maintenance is carried out only when required, leading to efficient use of resources.
- Allows for better spare parts management: Knowing in advance what parts might fail allows for better inventory management, reducing the need for overstocking and ensuring parts are available when needed.

- **Disadvantages of predictive maintenance**

By using predictive techniques, maintenance can be performed just in time to avoid unplanned downtime and improve equipment lifespan. While there are many advantages to this approach, there are also some disadvantages to consider:

- High initial costs: Setting up predictive maintenance typically requires investments in sensors, data analytics software, and sometimes even IoT (Internet of Things) infrastructure. For many companies, the upfront costs can be quite high.
- Complexity: Implementing predictive maintenance requires integrating different technologies and systems, analyzing vast amounts of data, and retraining personnel. This can introduce complexities that not every organization is equipped to handle.

• Over-reliance on technology: Just because the system predicts that a piece of equipment is fine doesn't always mean it is. There's always a risk of becoming too reliant on predictive data and ignoring other signs of equipment problems.

- **Real-time Alerts and Automation:**

Enhance the desktop application to provide real-time alerts not only for safety risks but also for operational inefficiencies (e.g., production bottlenecks, deviations from quality standards). Integrate with automation systems to enable immediate responses to alerts, such as adjusting production processes or dispatching maintenance teams.

Real-time alerts and automation systems in industrial environments are designed to enhance operational efficiency, safety, and responsiveness through the timely detection and immediate response to critical events. The underlying principles of effective real-time alerting and automation involve a blend of sensor technology, data processing capabilities, and decision-making algorithms. Firstly, sensors deployed throughout the factory continuously monitor various parameters such as temperature, pressure, and production metrics. These sensors generate real-time data streams that are transmitted to a central monitoring system or edge computing devices for rapid analysis. Secondly, advanced data processing techniques, including complex event processing (CEP) and predictive analytics, are employed to detect anomalies, deviations from optimal conditions, or potential hazards. These analyses are conducted in near real-time to ensure prompt detection and response. Thirdly, automated decision-making algorithms are integrated into the system to initiate immediate actions or alerts based on predefined rules or thresholds. This automa-

tion reduces dependency on human intervention for routine tasks, thereby improving response times and reducing the risk of human error. Furthermore, the integration of real-time alerting and automation with other systems, such as production scheduling or maintenance management software, enhances overall operational coordination and efficiency. By adhering to these principles, industrial facilities can achieve heightened situational awareness, proactive risk management, and streamlined operations, ultimately contributing to improved productivity and safety outcomes.

- **Integration of Edge Computing:** Implement edge computing techniques to process data closer to the source (e.g., on the STM32F103C8 and ESP32 devices). This reduces latency in data processing and enhances the system's responsiveness, critical for real-time monitoring and control in a smart factory environment.

In simplest terms, edge computing moves some portion of storage and compute resources out of the central data center and closer to the source of the data itself. Rather than transmitting raw data to a central data center for processing and analysis, that work is instead performed where the data is actually generated – whether that's a retail store, a factory floor, a sprawling utility or across a smart city. Only the result of that computing work at the edge, such as real-time business insights, equipment maintenance predictions or other actionable answers, is sent back to the main data center for review and other human interactions.

- **Scalability and Flexibility:**

Design the system architecture with scalability in mind, allowing easy integration of additional sensors, devices, and production

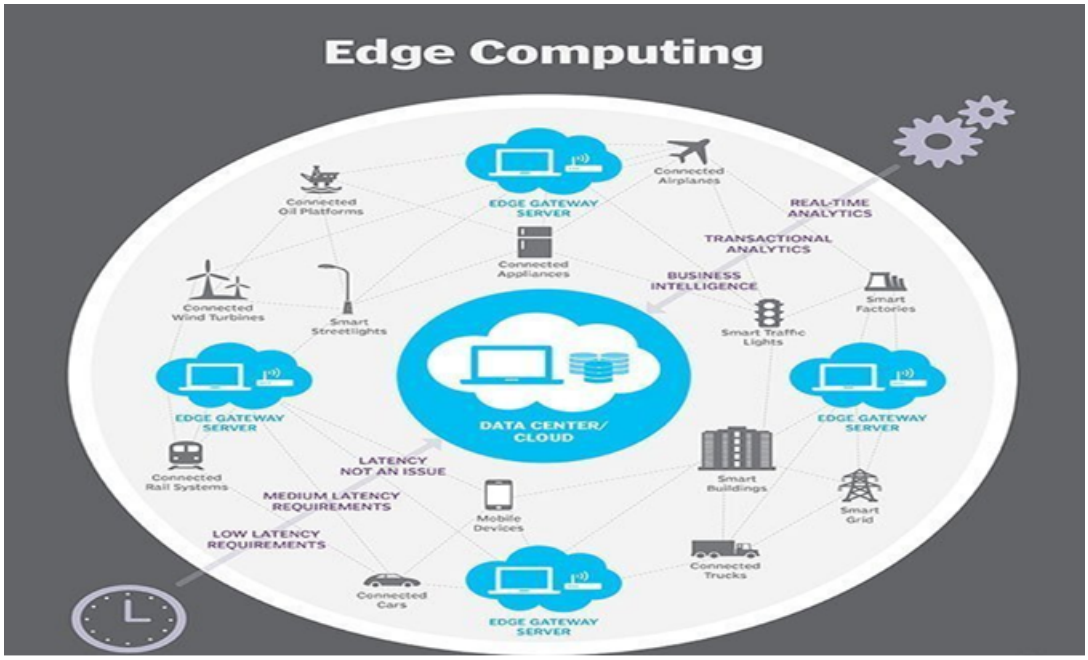

Figure 6.3: Diagram illustrating edge computing, showing how edge gateway servers enable real-time analytics and business intelligence by processing data from connected devices and systems locally, reducing latency and reliance on centralized data centers.

lines as the factory expands. Consider using modular, interoperable components and standardized communication protocols (e.g., MQTT) to facilitate future upgrades and expansions.

- **Data Analytics and Visualization:**

Develop advanced data analytics capabilities to derive actionable insights from the accumulated sensor and computer vision data. Implement interactive data visualization tools to present key performance indicators (KPIs) and trends, aiding in decision-making and process optimization.

Data analysis in industrial engineering involves using data to optimize processes, improve efficiency, and make informed decisions. Industrial engineers typically use data analysis in the following ways:

• Process Optimization: Analyzing production data to identify bottlenecks, inefficiencies, and opportunities for improvement in manufacturing processes.
• Quality Control: Using statistical analysis to monitor and maintain product quality, such as through Six Sigma methodologies.
• Lean Six Sigma: Lean and Six Sigma methodologies are widely used in industrial engineering for process improvement. They rely heavily on data analysis to reduce defects, variability, and waste.
• Operations Research: Industrial engineers use operations research techniques to optimize complex systems, like supply chains and logistics. Linear programming, network analysis, and queuing theory are common tools in this domain.
• Time and Motion Studies: Analyzing data on worker movements and time spent on tasks to enhance workplace ergonomics and efficiency.
• Inventory Management: Employing data analysis to optimize inventory levels, reduce carrying costs, and ensure timely supply chain management.
• Simulation and Modeling: Creating computer models and simulations to predict how changes in processes or systems will impact productivity and cost.
• Cost Analysis: Analyzing cost data to identify cost-saving opportunities and make budgetary decisions.
• Supply Chain Optimization: Using data to improve the flow of materials, reduce lead times, and enhance supplier relationships.
• Energy Efficiency: Analyzing energy consumption data to reduce energy costs and environmental impact.
• Risk Assessment: Using data to assess and mitigate risks in industrial processes, safety, and project management.
• Data-Driven Decision-Making: Making informed decisions based on data analysis results to drive continuous improvement in indus-

trial processes.

• Cost-Benefit Analysis: Evaluating the cost-effectiveness of process changes and investments is a crucial aspect of data analysis, especially in making decisions about capital expenditures.

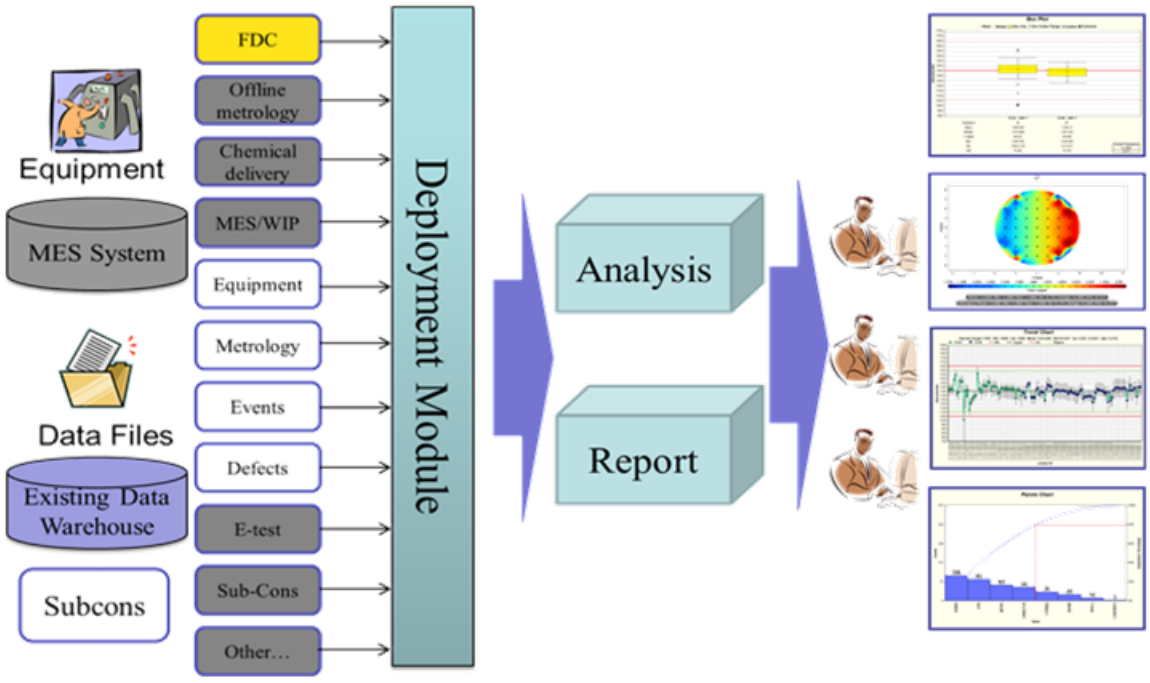

Figure 6.4: Data Deployment and Analysis Workflow

- **Energy Efficiency:**

Optimize the power consumption of IoT devices and sensors to prolong battery life and reduce operational costs. Implement energy harvesting techniques where feasible, such as using solar panels or kinetic energy converters to power low-energy sensors.

Energy efficiency is a critical concern in modern industrial and technological contexts, driven by principles aimed at reducing environmental impact and operational costs while optimizing resource utilization. The fundamental principles of energy efficiency encompass technological innovation, operational practices, and sustain-

ability goals. Technologically, advancements in equipment design, such as the development of energy-efficient motors, sensors, and heating/cooling systems, play a crucial role in minimizing energy consumption during production processes. These technologies are often complemented by intelligent control systems that optimize energy usage based on real-time demand and environmental conditions. Operational practices emphasize the importance of monitoring and optimizing energy-intensive processes through data-driven analytics and continuous improvement initiatives. Implementing energy management systems that track and analyze energy usage patterns enables organizations to identify inefficiencies and implement targeted interventions to achieve energy savings. Furthermore, adherence to sustainability goals involves integrating renewable energy sources, such as solar and wind power, into industrial operations to reduce reliance on fossil fuels and mitigate carbon emissions. By integrating these principles into comprehensive energy management strategies, organizations can achieve substantial energy savings, enhance operational resilience, and contribute to broader environmental sustainability objectives in the industrial sector.

- **Cybersecurity Measures:**

Strengthen cybersecurity measures across the entire system, from IoT devices to the desktop application. Implement encryption protocols for data transmission, secure authentication mechanisms, and regular security audits to protect against potential cyber threats and unauthorized access.

Cybersecurity measures constitute a foundational element in safeguarding digital assets and maintaining operational integrity across various sectors, including industrial environments. The

principles underpinning effective cybersecurity systems are multifaceted and encompass both proactive defenses and reactive strategies. Proactively, robust cybersecurity begins with comprehensive risk assessment and threat modeling to identify potential vulnerabilities within industrial systems. This process informs the implementation of defense-in-depth strategies, which involve layers of security controls such as firewalls, intrusion detection/prevention systems, and network segmentation to mitigate risks and limit the impact of potential breaches. Furthermore, adherence to cybersecurity principles involves continuous monitoring of network traffic and system activity to promptly detect and respond to suspicious behaviors or anomalies. Reactive measures include incident response plans and procedures that outline clear steps for containment, eradication, and recovery in the event of a cyber incident. These plans often involve cross-functional collaboration between IT professionals, operational teams, and executive leadership to minimize disruption and ensure rapid restoration of normal operations. Additionally, cybersecurity principles emphasize the importance of ongoing education and training for personnel at all levels to cultivate a culture of vigilance and accountability in cybersecurity practices. By integrating these principles into holistic cybersecurity frameworks, industrial organizations can effectively mitigate cyber threats, protect sensitive data, and uphold operational resilience in an increasingly interconnected and digital landscape.

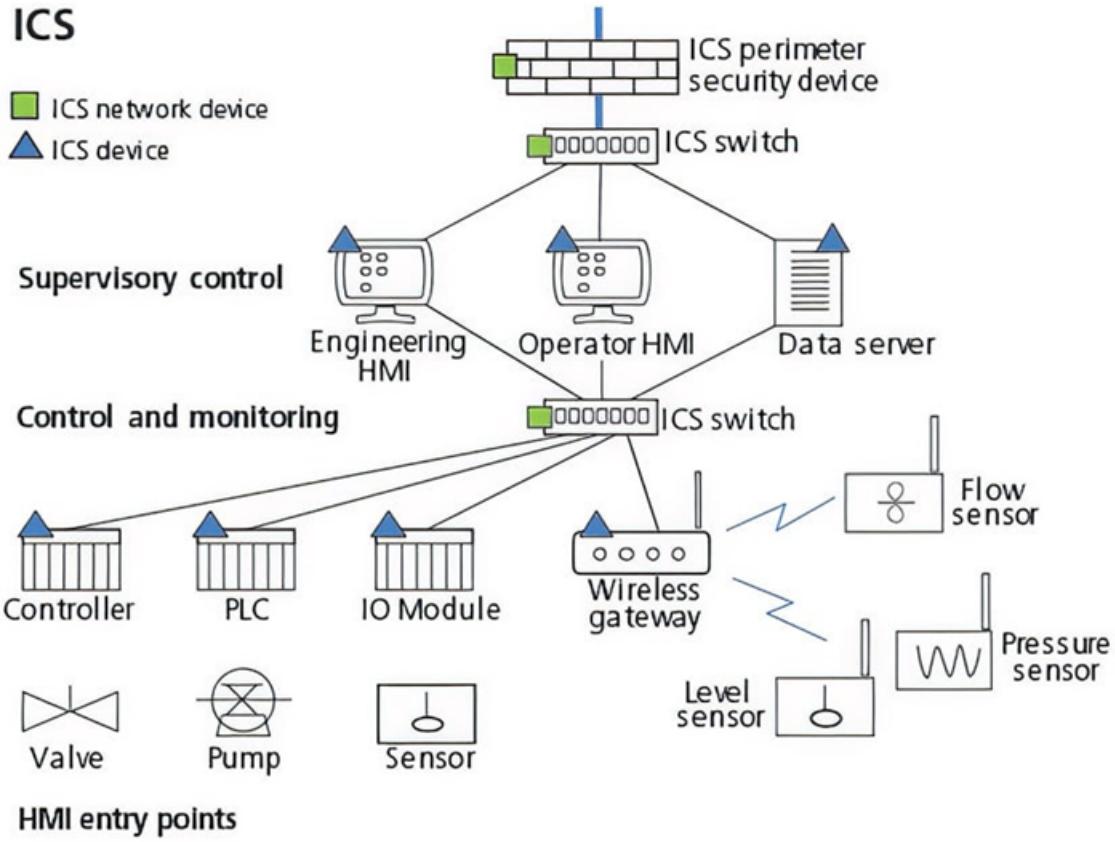

Figure 6.5: Industrial Control System (ICS) Architecture

Chapter 7

References

- Semtech Corporation, *SX1278 Datasheet*, 2017. Available: `https://www.semtech.com/uploads/documents/sx1278.pdf`. [Accessed: 2024-06-22].
- STMicroelectronics, *STM32H7 Datasheet*, 2020. Available: `https://www.st.com/resource/en/datasheet/stm32h7.pdf`. [Accessed: 2024-06-22].
- Joseph Redmon and Ali Farhadi, *YOLO Algorithm Documentation*, 2018. Available: `https://pjreddie.com/media/files/papers/YOLOv3.pdf`. [Accessed: 2024-06-22].
- Adafruit Industries, *DHT22 Datasheet*, 2015. Available: `https://cdn-shop.adafruit.com/datasheets/DHT22.pdf`. [Accessed: 2024-06-22].
- OmniVision Technologies, *OV5640 Datasheet*, 2014. Available: `https://www.ovt.com/uploads/parts/OV5640.pdf`. [Accessed: 2024-06-22].