AI-based Obstacle Detection for Autonomous Train Control using Image Recognition

Lorenz Maria Tiedemann

Bachelor's thesis
March 2025

Prof. Reinhard von Hanxleden
Real-Time and Embedded Systems Group
Department of Computer Science
Kiel University

Advised by
Dr.-Ing. Alexander Schulz-Rosengarten
M.sc. Momin Ali

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Weiterhin erkläre ich, dass die digitale Fassung dieser Arbeit, die dem Prüfungsamt per E-Mail zugegangen ist, der vorliegenden schriftlichen Fassung entspricht.

Kiel,

Abstract

Deep learning-based vision systems have transformed autonomous driving in the automotive sector, and their potential is now being explored for rail traffic. In this thesis, we present a camera-based obstacle detection system that supports autonomous rail operations. Our work focuses on training and testing a deep neural network model that reliably identifies obstacles along the tracks. In addition, we implement an innovative scoring system that communicates directly with the autonomous controller, converting visual detections into actionable inputs for automated driving decisions. Experimental evaluations demonstrate that the integrated model and scoring mechanism effectively facilitate autonomous rail operation.

Acknowledgements

I want to begin by thanking Prof. Dr. Reinhard von Hanxleden, Head of the Real-Time and Embedded Systems Group, who made this thesis possible. I am also very grateful to my supervisors, Dr.-Ing. Alexander Schulz-Rosengarten and Momin Ali, for their guidance and valuable feedback throughout this thesis. My thanks go to Sven Ratjens for providing access to the Malente Lütjenburg railway track for outdoor data collection and testing, as well as to Birkan Deninzer for introducing me to the NVIDIA Jetson devices and to Gerd Diesner for his assistance with the training server. Finally, I appreciate the excellent collaboration with Kevin Ebsen, Nig Rambow, Simon Jürgensen, and Rasmus Janssen.

Contents

1 Introduction **3**
- 1.1 Problem Statement . . . 4
- 1.2 Outline . . . 4

2 Tools and Technologies **5**
- 2.1 Convolutional Neural Networks . . . 5
- 2.2 Vision Transformers . . . 5
- 2.3 Ultralytics YOLO . . . 8
- 2.4 RT-DETR . . . 8
- 2.5 Python . . . 9
- 2.6 OpenCV . . . 9

3 Related Work **11**
- 3.1 Automotive Perspective . . . 11
- 3.2 Recent Research . . . 11
- 3.3 Sensor Fusion . . . 12
- 3.4 Industry Applications . . . 13
- 3.5 Additional Projects and Integration with Our Concept . . . 13

4 Concept **15**
- 4.1 System Overview . . . 15
- 4.2 Rail Segmentation . . . 16
  - 4.2.1 Motivation and Alternatives . . . 16
  - 4.2.2 Reasons for Our Choice . . . 17
- 4.3 Obstacle Recognition . . . 17
  - 4.3.1 Motivation and Alternatives . . . 17
  - 4.3.2 Reasons for Our Choice . . . 18
- 4.4 Scoring . . . 18
  - 4.4.1 Purpose and Structure . . . 18
  - 4.4.2 Scoring Details . . . 19
  - 4.4.3 Thresholds and Tunable Variables . . . 20
- 4.5 Summary . . . 20

5 Implementation **21**
- 5.1 Data Preparation and Model Training . . . 21
- 5.2 Real-Time Pipeline . . . 23
- 5.3 Scoring and Irregularity Detection . . . 24

5.3.1 Rail Score . . . 24
5.3.2 Obstacle Score . . . 24
5.3.3 Overall Score . . . 25
5.4 Multi-Threaded Integration . . . 25
5.5 Annotated Output . . . 25
5.6 Thresholds and Tunable Parameters . . . 25
5.7 Summary . . . 26

6 Evaluation **27**
6.1 Understanding the Metrics and Diagrams . . . 27
6.2 Confusion Matrix Analysis . . . 27
6.3 Bounding Box Detection Metrics . . . 28
6.4 Segmentation Mask Metrics . . . 31
6.5 Pipeline Speed . . . 34
6.6 Field Test Results . . . 34
6.7 Discussion and Implications . . . 36
6.8 Summary . . . 37

7 Conclusion **39**
7.1 System Capabilities and Production Readiness . . . 39
7.2 Future Work . . . 39

Bibliography **41**

List of Figures

1.1 Single-Track Transfer Traffic (©Muthesius Kunsthochschule) 3

2.1 Schematic diagram of a convolutional neural network (CNN). [1] 6
2.2 Schematic architecture of a Vision Transformer (ViT). [2] 7

3.1 Detection Example[3] 14

4.1 System concept diagram. 16

5.1 Pipeline overview. 22
5.2 Module placement. 23
5.3 Detecting irregularities. 24
5.4 Annotated detection. 25

6.1 Normalized confusion matrix for rail segmentation, showing about 87% correct classification of rail pixels. 28
6.2 Rail bounding box precision-recall curve. At a threshold around 0.8, the magenta curve (rail) shows a precision of about 83% and recall near 88%. . . . 29
6.3 Rail bounding box F1 curve, with scores around 84% for the magenta rail curve. 29
6.4 Rail bounding box precision curve showing how precision varies with the confidence threshold (magenta: rail). 30
6.5 Rail bounding box recall curve showing how recall varies with the confidence threshold (magenta: rail). 31
6.6 Rail segmentation mask precision-recall curve, with precision near 95% and recall around 90% (magenta: rail). 32
6.7 Rail segmentation mask F1 curve, with scores around 80% (magenta: rail). . . 32
6.8 Rail segmentation mask precision curve (magenta indicates the rail performance). 33
6.9 Rail segmentation mask recall curve (magenta indicates the rail performance). 33
6.10 Inference speed measured in Frames Per Second (FPS). Our system processes about 9 FPS, corresponding to a per-frame delay of approximately 111 ms. . . . 34
6.11 Field Test: Successful detection example. The system accurately identifies the rail marked green and obstacles marked blue, and orange, with a rail score of approximately 43 and an obstacle score of 80. 35

[1] Source: Perelló Nieto, CC BY 4.0.
[2] Source: Daniel Voigt Godoy, CC BY 4.0.
[3] Source: Rail Vision Ltd. – Obstacle Detection Demo (© Rail Vision, 2024).

6.12 Field Test: Detection failure case. Poor image quality due to vibrations leads to an overall score of about 20, indicating missed detections under challenging conditions. 36

List of Tables

List of Acronyms

CNN Convolutional Neural Network

ViT Vision Transformer

YOLO You Only Look Once

RT-DETR Real-Time Detection Transformer

DETR Detection Transformer

NMS Non-Maximum Suppression

OpenCV Open Source Computer Vision Library

AI Artificial Intelligence

LiDAR Light Detection and Ranging

COCO Common Objects in Context

RTSP Real Time Streaming Protocol

IR Infrared

FPS Frames Per Second

Chapter 1

Introduction

Autonomous rail traffic is increasingly recognized as a promising solution for modern transportation challenges. With the persistent shortage of qualified train drivers and the pressing need to reduce operational costs, automating rail operations has become essential. Automation can improve efficiency by handling routine tasks such as obstacle detection and navigation, ultimately paving the way for fully driverless trains. This approach not only benefits conventional rail services but also enables innovative on-demand operations.

A compelling example of this vision is the concept of Single-Track Transfer Traffic, as illustrated in Figure 1.1. In this model, autonomous vehicles travel on a single track and, when meeting from opposite directions, briefly dock to exchange passengers before proceeding in the new direction. Known as Single-Track Transfer Traffic, this concept optimizes track usage and showcases the potential for flexible rail services. Projects like REAKTOR leverage this idea to deploy autonomous trains on underutilized or decommissioned tracks, thereby offering efficient on-demand rail transport.

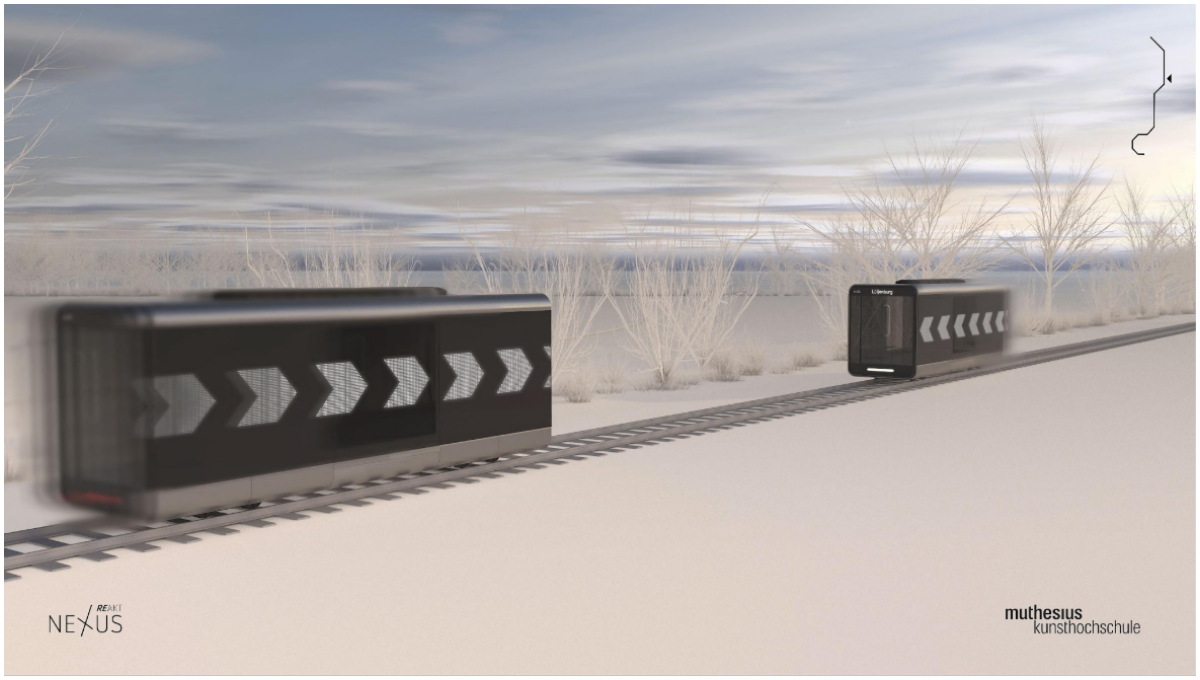

Figure 1.1. Single-Track Transfer Traffic (©Muthesius Kunsthochschule)

1.1 Problem Statement

The rail industry faces a significant shortage of qualified train drivers, which necessitates the automation of train operations. A critical component of this automation is a reliable obstacle detection system that can accurately perceive the environment and communicate with the onboard autonomous controller. In this thesis, we address this challenge by training a deep neural network model to detect obstacles on the tracks and implementing an innovative scoring system that conveys essential information to the autonomous controller. This work lays the foundation for safe and efficient autonomous rail operations.

1.2 Outline

This thesis first reviews the fundamental tools and technologies that underpin modern AI-driven obstacle detection and then discusses related work, focusing on the current state of research in both the automotive and railway domains. It proceeds by introducing a conceptual framework for a two-stage solution that handles rail segmentation and obstacle recognition separately, followed by an in-depth examination of the implementation details and the data preparation pipeline. An evaluation section then presents key findings and performance metrics, including an analysis of limitations and potential improvements. Finally, the thesis concludes by summarizing the main contributions and suggesting areas for future research, especially in relation to sensor fusion, multi-object tracking, and hardware optimization.

Chapter 2

Tools and Technologies

These days, Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) form the foundation of most modern computer vision tasks, including obstacle detection. Specific models such as You Only Look Once (YOLO) and Real-Time Detection Transformer (RT-DETR), the programming language Python, and the open source library Open Source Computer Vision Library (OpenCV) serve as fundamental building blocks in research and practice. In the following, we introduce and explain these tools and technologies.

2.1 Convolutional Neural Networks

Convolutional neural networks are artificial neural networks that researchers design specifically for image processing. They use convolutional kernels to extract the characteristics and features of local image sections. Yann LeCun et al. [LBB+98] introduce this concept at the end of the 1980s and use it successfully in the 1990s to recognize handwritten numbers. In 2012, researchers achieve a crucial breakthrough when a deep CNN wins the ImageNet contest with a big lead. At that moment, CNNs become the standard approach for image classification, object detection, and other tasks.

As illustrated in Figure 2.1, a CNN typically consists of multiple convolutional, pooling, and fully connected layers stacked in series. This diagram shows how the network processes local image features at each stage to learn increasingly complex representations.

Their deep structure allows them to learn hierarchical characteristics, from simple edges in early layers to whole object parts in deeper layers. Optimization techniques such as batch normalization and dropouts stabilize the training of deep networks and reduce overfitting. CNNs generalize on big datasets while keeping a low inference time.

Inspired by the successes in automotive applications, our concept adapts these robust CNN architectures to segment rail environments, ensuring precise identification of rail areas for further processing.

2.2 Vision Transformers

Vision Transformers transfer the concept of the transformer architecture—originally developed for language processing—to the domain of image processing. Unlike CNNs, which use convolutional layers, ViTs treat images as a sequence of patch representations. Figure 2.2 shows the schematic architecture of a Vision Transformer (ViT). The input image is separated into

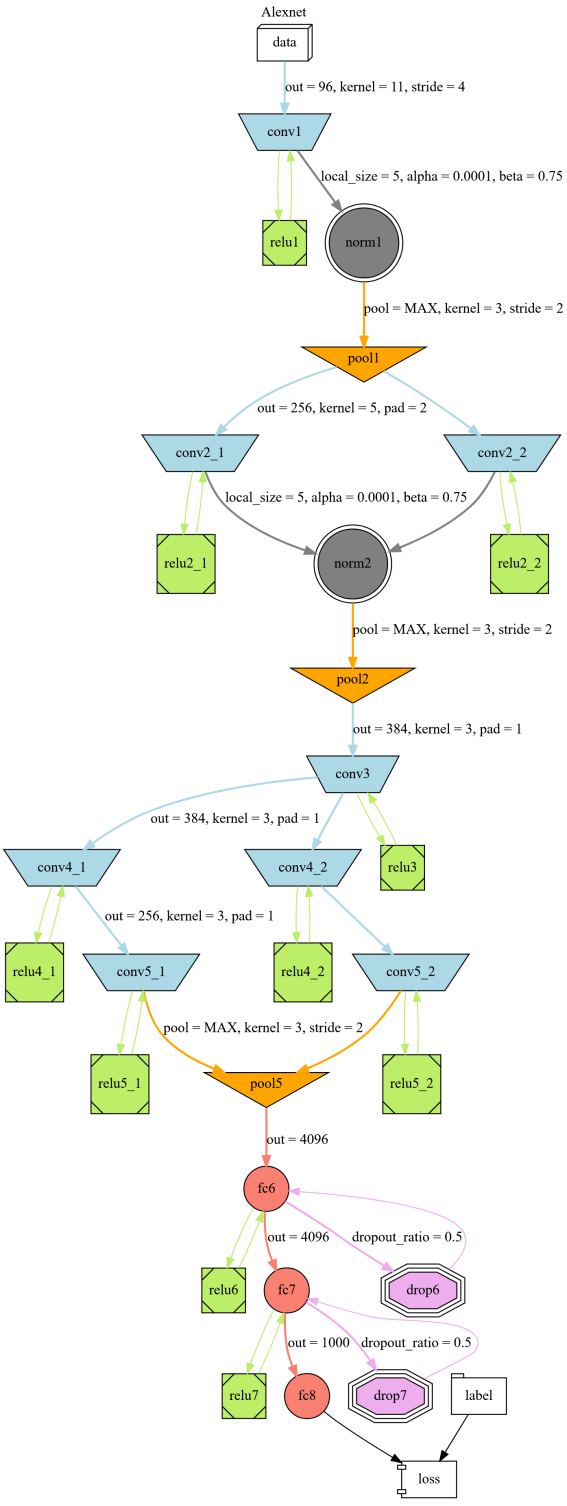

Figure 2.1. Schematic diagram of a convolutional neural network (CNN). [a]

[a] Source: Perelló Nieto, CC BY 4.0.

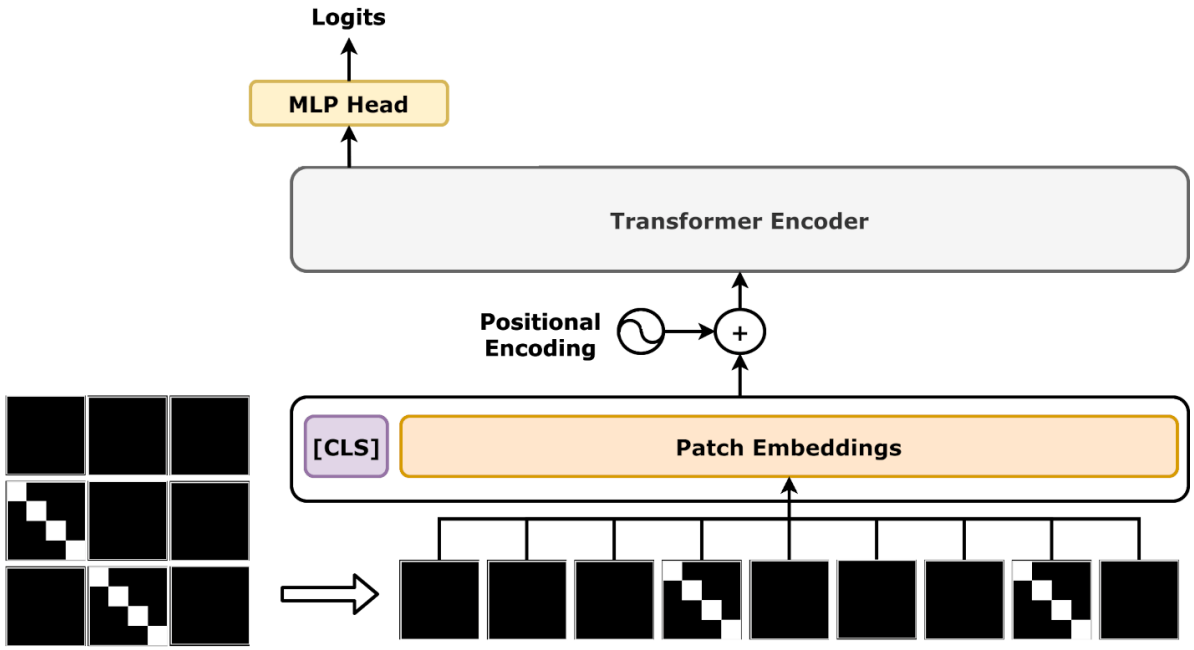

Figure 2.2. Schematic architecture of a Vision Transformer (ViT). [a]

[a]Source: Daniel Voigt Godoy, CC BY 4.0.

small, equally sized patches that are used as a sequence for the transformer input, analogous to words in a sentence [DBK+21]. First it embeds each patch in a feature vector using a linear projection and supplies it with positional information. Then a transformer encoder with a self-attention mechanism models global dependencies between all image regions. The decisive advantage of ViTs lies in their capability to capture global context. While CNNs process only local connections due to their limited kernel size, transformer-based models incorporate information from the entire image. This enables ViTs, especially on large datasets, to achieve results equal to or better than CNNs [DBK+21].
Our approach benefits from the global context captured by ViTs to complement the localized segmentation performed by CNNs, thus enhancing the overall robustness of our obstacle detection system.

2.3 Ultralytics YOLO

YOLO stands as a groundbreaking approach for real-time object detection; Joseph Redmon et al. [RDG+16] propose it in 2016. In contrast to older two-stage models that first determine a region of an object and then classify it, YOLO uses a single-stage approach for object detection. YOLO is a CNNs-based model; it divides the input image into a grid and determines, for each cell, the class probability and the bounding boxes. Due to its end-to-end design, YOLO achieves high frame rates, making it suitable for time-critical tasks.

Ultralytics YOLO relates to the company Ultralytics, which develops and maintains a series of YOLO implementations—in particular YOLOv5, YOLOv8, and YOLOv11. These models build on the original YOLO design while incorporating numerous architectural and training improvements. This includes an advanced backbone architecture, more efficient Feature Pyramid structures for object detection at different scales, techniques such as mosaic data augmentation, and automatic anchor generation. Ultralytics provides pre-trained models within a user-friendly Python library that simplifies research and development. Although Ultralytics YOLO does not appear in a conventional scientific paper, practitioners regard it as a powerful representative of current single-stage detectors and widely use it in areas such as traffic monitoring, robotics, and augmented reality [JQ24].

For our concept, we use YOLO not only for object detection but also for segmenting the rail area, thereby creating a focused region for subsequent obstacle detection.

2.4 RT-DETR

RT-DETR is an object detection model that utilizes a transformer architecture and enables real-time usage. Zhao et al. [ZLX+24] present RT-DETR in 2024 under the title "DETRs Beat YOLOs on Real-Time Object Detection." The model is tied to the Detection Transformer (DETR) architecture. In 2020, DETR becomes the first end-to-end object detection model that utilizes a transformer architecture. DETR demonstrates that a heuristic Non-Maximum Suppression (NMS) step is not required when the model predicts a fixed number of objects through direct set matching. However, DETR runs slowly and remains inefficient for higher resolution images, which does not suit time-critical applications.

RT-DETR addresses this problem and performs real-time tasks as its name implies. It uses an efficient hybrid transformer encoder to process multiscale features of the image in parallel and an optimized decoder with a special query selection algorithm. The query selection algorithm selects the initial query points for objects in a way that maximizes detection accuracy. This means RT-DETR does not require an NMS step and matches the inference time of YOLO models. Zhao et al. [ZLX+24] report that RT-DETR achieves the same precision as contemporary YOLO models while delivering a higher frame rate. The Ultralytics library implements RT-DETR, which equals the user experience of YOLO and confirms the shift of computer vision models to ViTs architectures.

In our implementation, RT-DETR plays a key role in detecting obstacles within the seg-

mented rail region, ensuring rapid and accurate identification of potential hazards.

2.5 Python

Python is a universally interpreted programming language that is well established for big data, machine learning tasks, and many other applications. It is well known for its extensive software libraries and easy-to-read syntax, which allow fast prototyping. Frameworks like TensorFlow and PyTorch, as well as libraries like Ultralytics, enable us to define complex neural networks and train them in only a few lines of code.

The major advantage of Python is its extensive ecosystem, which includes packages such as NumPy and Pandas for numerical calculations and data analysis, as well as SciPy and Scikit-learn for statistics and machine learning. Python itself runs slower than compiled languages like Rust or C because it is interpreted, but it overcomes this drawback by integrating high-performance libraries. Developers implement computationally intensive tasks in C/C++ or CUDA within these libraries and run them in a highly optimized manner while Python orchestrates all the different functions. This is the reason why Python fully utilizes the hardware potential of modern systems, such as graphics processing units, multicore processors, and tensor processing units – specialized hardware accelerators for Artificial Intelligence (AI) tasks. The simplicity of Python promotes the reproducibility of research results because the code remains easy to share and understand. These attributes make Python the preferred tool in the deep learning community.

Our concept benefits from Python's flexibility, enabling rapid prototyping and seamless integration of our deep learning pipelines.

2.6 OpenCV

OpenCV is a free software library for computer vision, image manipulation, and processing tasks. Intel initiates OpenCV and publishes it in 2000 [Bra00]. OpenCV contains well over 100 implementations of fundamental functions for image processing and machine learning, including image filtering, geometrical transformations, feature extraction, and tracking. Developed in C/C++, OpenCV performs very well and provides interfaces for many languages, including Python, which makes it suitable for use with modern deep learning libraries.

In practice, practitioners often use OpenCV in combination with CNNs and ViTs for pre- and post-processing tasks such as loading input images, extracting individual images from a video stream, scaling the image, or normalizing it before the object detection model processes it. After inference, one can use OpenCV to display detection results, for example, by drawing a bounding box around a detected object and combining individual images into a video stream. Through its open license and large developer community, OpenCV becomes the go-to library for image processing.

For our concept, OpenCV is essential for real-time visualization of the detection results, enabling quick validation and debugging of our system.

Chapter 3

Related Work

RailSem19 stands as a cornerstone dataset in the field of image-based obstacle detection on rails. Zendel et al. [ZMZ+19] introduce the RailSem19 dataset in 2019 as the first publicly available dataset for semantic scene understanding in the rail domain. It comprises approximately 8,500 ego perspective images captured from trains and trams, with comprehensive pixel-wise annotations of rail-specific classes (e.g., tracks, switches, signals) that conventional road datasets do not include. By providing dense semantic labels for a wide range of rail infrastructure and surrounding objects, RailSem19 fills a crucial data gap and enables researchers to train and evaluate computer vision models tailored for railway scenes [ZMZ+19]. Our concept directly leverages this dataset to train models that accurately segment the rail environment, for obstacle detection we leverage the Common Objects in Context (COCO) dataset [LMB+14].

3.1 Automotive Perspective

Large-scale datasets and significant research efforts support camera-based perception for road vehicles. Modern cars and autonomous driving systems rely on various sensors, such as cameras, Light Detection and Rangings (LiDARs), and radar, and use AI algorithms to evaluate sensor data in order to detect vehicles, pedestrians, obstacles, and hazards with high reliability. Publicly available benchmarks like KITTI and Cityscapes accelerate the development of deep learning models capable of accurate object detection and semantic segmentation in complex road scenes [MTM12]. These models, such as CNNs, learn to recognize obstacles and infrastructure from vast amounts of annotated images. The success in automotive perception—such as detecting a car, a pedestrian, or other objects in various lighting and weather conditions—offers valuable insights. Our concept adapts these proven techniques to the rail domain by tailoring them to the unique challenges of railway environments, ensuring both high accuracy and real-time performance.

3.2 Recent Research

In recent years, research on obstacle detection for railways gains momentum as the field shifts from traditional image processing to data-driven AI approaches [RAH+22]. A comprehensive review by Ristić-Durrant et al. [RFM21] notes that although railway applications lag behind, the community rapidly adopts modern computer vision techniques. Several notable studies

emerge. Ye et al. [YWS+18] develop deep learning-based systems for onboard obstacle detection in front of trains. In their work, they propose a Feature Fusion Refine Neural Network (FR-Net) to identify objects on or near the tracks during shunting operations. Their approach fuses visual features at multiple scales, enabling the detection of various obstacle types—including locomotives, track workers, and track conditions (e.g., rail misalignment). The system pairs with a millimeter wave radar to measure the distance to detected objects, enhancing safety by providing both recognition and range estimation. This vision and radar system achieves high precision in detecting six classes of railway obstacles, from large trains to small items like a fallen helmet, and alerts train operators in real time [YWS+18]. The same authors later report an improved model with differential feature fusion, demonstrating continued progress in detection accuracy and robustness [YWS+20]. In our concept, we build upon these advancements by integrating state-of-the-art CNN and transformer-based models, fine-tuned specifically for rail environments, ensuring that the latest research developments are directly applied to our system.

3.3 Sensor Fusion

Another important aspect of railway obstacle detection research involves using alternative sensors and sensor fusion to complement camera-based AI. While cameras provide high-resolution visual information, they suffer under challenging lighting or weather conditions (e.g., rain, snow, darkness, fog), where other sensors perform better [GVP+20]. To leverage the strengths of each modality, researchers increasingly adopt multi-sensor setups [GVP+20]. Researchers have demonstrated obstacle detection systems that combine video cameras with active sensors such as radar, LiDAR, or thermal-infrared cameras [GVP+20]. For instance, one experimental system pairs an infrared camera with a laser rangefinder to detect obstacles on tracks even in low-visibility conditions [TKT+23]. Others test the fusion of a forward-facing optical camera with a long-range radar, achieving reliable detection of track obstructions by merging visual recognition with precise distance measurements [TKT+23]. Gleichauf et al. (2020) present a multi-sensor approach for an autonomous shunting locomotive that integrates multiple LiDAR units (including a short-range 3D laser scanner and a long-range scanning LiDAR) alongside optical cameras [GVP+20]. By fusing LiDAR's accurate depth sensing with camera imagery, their system detects and localizes obstacles, and even classifies objects such as wagons over a broad range in front of a train [GVP+20]. Another prototype combines a regular RGB camera with a thermal-imaging camera and achieves more robust obstacle detection in nighttime and foggy scenarios by capturing heat signatures that visible-spectrum cameras might miss [TKT+23]. These research efforts underscore that image-based AI greatly benefits from additional sensor data in safety-critical rail applications. Meanwhile, vision remains central because cameras capture fine detail and color information that radar or LiDAR cannot, enabling the classification of obstacle types (e.g., animal, person, vehicle) [MTM12]. As Silla and Künhert's experiments highlight, only optical sensors provide the necessary context to distinguish whether an object lies on the track or nearby, which is vital for decision-making.

Although sensor fusion is a promising approach, it lies outside the scope of this work; we deliberately rely on a single-camera solution because it is significantly more cost-effective and simpler to deploy.

3.4 Industry Applications

Industry developments mirror these research trends as railway companies implement obstacle detection in real-world operations. For example, Alstom tests an obstacle detection system (ODS) for autonomous freight trains in the Netherlands. Their ODS, developed with Elta Systems/NIART, uses a high-resolution radar combined with electro-optical cameras and onboard AI to detect obstacles up to 1000 m ahead under all weather and lighting conditions. This system functions as a driver assistance tool that warns human operators of hazards and prepares for full integration with autonomous driving (GoA4) capabilities. The tests demonstrate the detection of both large objects (e.g., a car on the tracks) and smaller ones like animals or fallen debris at long range, day or night. They achieve such range and robustness by fusing radar with camera sensors for classification. Similarly, Siemens Mobility trials an obstacle detection system on a commuter S-Bahn train in Berlin [Sie24]. In 2024, Siemens equips an S-Bahn train with a suite of sensors and AI as part of the "Digital Rail Germany" initiative, marking the first use of an obstacle detection setup in daily passenger service. During these trials, the system continuously records and analyzes the tracks ahead. This data validates performance across varying weather and lighting conditions and optimize sensor placement on the train. Our concept is directly aligned with these industrial applications by providing a scalable, cost-efficient, and accurate AI-based obstacle detection method that can be integrated into existing rail systems, and supporting autonomous operation.

3.5 Additional Projects and Integration with Our Concept

Additional projects further demonstrate the industry trend towards AI-based obstacle detection and autonomous train operation. Notably, projects such as Raileyes [EYY24], C3CAM [Bea24], and Futurail [Fut24] adopt similar approaches by integrating deep learning to achieve reliable obstacle detection in railway environments. These projects share common design principles with our concept and implementation, thereby reinforcing the validity and relevance of our work. Figure 3.1 shows two pedestrians on a railway track, each enclosed by a bounding box to indicate a potential obstacle for autonomous detection. This example underscores how modern AI systems visually mark individuals or objects that intrude onto the rails.

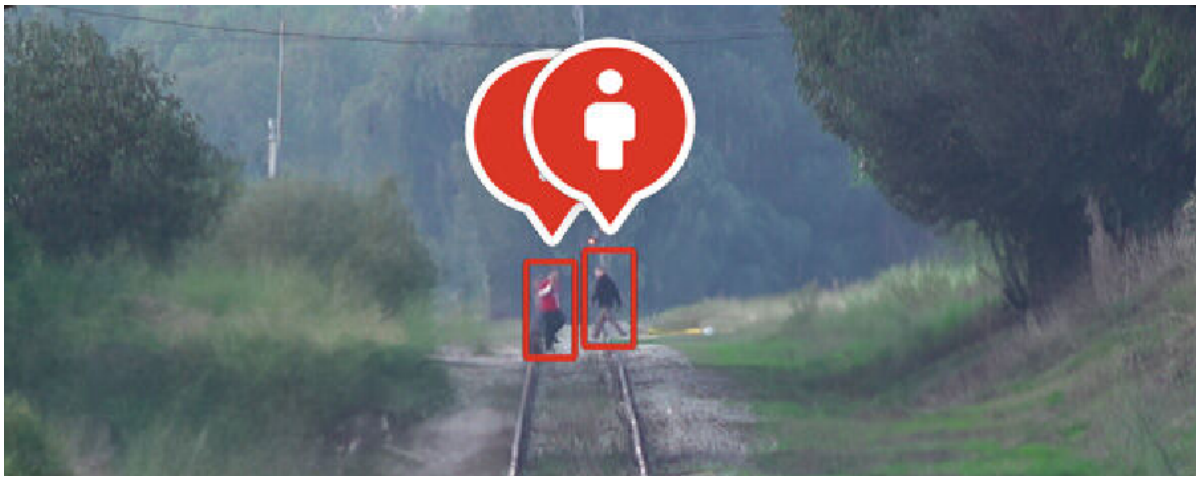

Figure 3.1. Detection Example[a]

[a]Source: Rail Vision Ltd. – Obstacle Detection Demo (© Rail Vision, 2024).

Chapter 4

Concept

In this chapter, we introduce our camera-based system for detecting obstacles on rail tracks. We structure the concept into three key components: rail segmentation 4.2, obstacle recognition 4.3, and scoring 4.4. We describe each step, explain why we need it, and discuss the advantages and disadvantages of our design decisions.

4.1 System Overview

Our goal is to enable autonomous train operation by providing a reliable perception pipeline that identifies rail tracks and detects any obstacles on them in real time. To achieve this, we divide the task into two vision-based modules: one dedicated to accurately segmenting the rail ("Rail Segmentation"), and another specialized in detecting potential obstacles ("Obstacle Recognition"). Finally, we merge both outputs in a scoring system that supplies a single, numeric indicator ("Overall Score") to an autonomous controller.

Figure 4.1 shows how the components interact with each other. On the top left, the system receives a video signal and uses a rail segmentation module to highlight the track region. On the top right, an object detection module identifies potential obstacles within the image. The outputs of these modules are then scored by two separated scoring modules. At the bottom, an overall scoring module merges the outputs into a single, easy-to-read metric.

This modular design offers several benefits. We separate concerns by handling rail detection and obstacle detection independently, which simplifies the design and facilitates targeted improvements for each module. The system achieves a degree of flexibility through this design, which makes it possible to swap out or fine-tune the rail segmentation model, obstacle detector, or scoring modules without breaking the entire pipeline. This flexibility is essential when new training data becomes available so that a newly trained model can be swapped in without rebuilding the whole system. In addition, we can tweak the scoring system without modifying the detection and segmentation modules. The system evaluates the final output with a unified scoring mechanism that accounts for both track continuity and obstacle proximity, thereby offering the advantage of robustness; for example, if the obstacle detector fails to identify an obstacle but the rail segmentation reveals a discontinuity in the track, the overall score still increases.

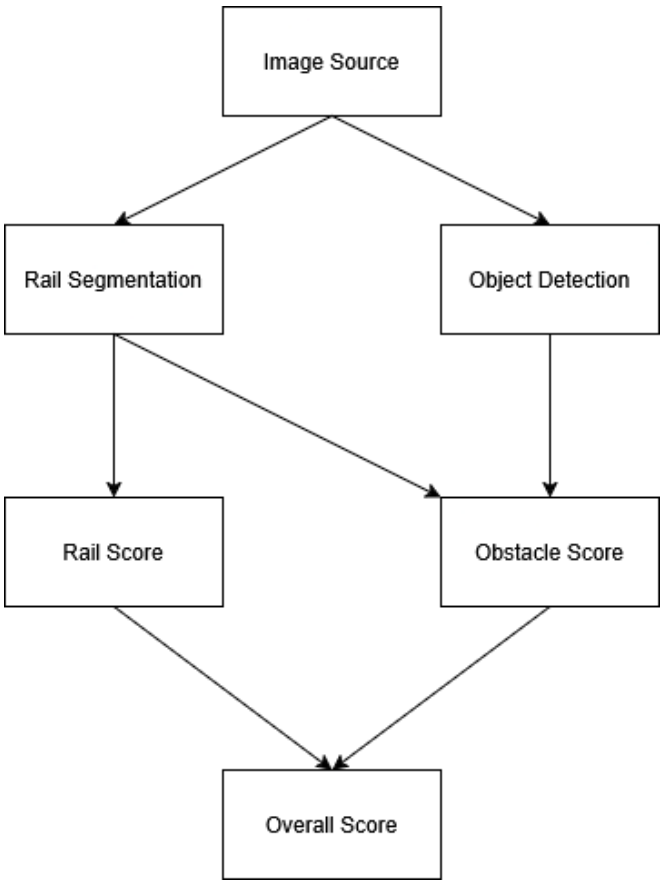

Figure 4.1. System concept diagram.

4.2 Rail Segmentation

4.2.1 Motivation and Alternatives

We segment the rail to determine where the train can move safely and to focus subsequent obstacle detection on relevant track areas. This step is essential because our controller must know whether an object lies *on* the rails or merely near them. A reliable rail segmentation model, therefore, becomes the foundation for evaluating which obstacles pose an immediate threat.

We consider several approaches before deciding on a dedicated rail segmentation network:

- **Classical Image Processing:** Traditional methods rely on edge detection or color thresholds. However, these hand-crafted approaches struggle with environmental variations (e.g., changing lighting, rail conditions) and do not generalize well to new track types.
- **Multi-Class Rail-and-Obstacle Segmentation:** We could train a single model to label both rails and obstacles in one pass. This approach simplifies the pipeline, but it requires

extensive manual annotation of obstacles for every scene. It also tends to increase model size and training complexity.

- ▻ **Dedicated Rail Segmentation:** When a model focuses solely on one dedicated task, we can train it specifically for that task, and building a training set for a dedicated task involves less work than building one for multiple tasks at once.

After comparing these alternatives, we opt for a dedicated, single-class segmentation model for the rail. This approach focuses on the track area with high accuracy and runs fast enough to support real-time inference. We base the model on a deep neural network architecture (e.g., YOLO or a similar segmentation backbone) and train it on rail-specific datasets (e.g., RailSem19). By restricting the task to one class, we reduce annotation overhead and maximize reliability for the rail region.

4.2.2 Reasons for Our Choice

Accurate rail segmentation is critical for assessing obstacles in a rail-centric context. It directly answers the question of *where* the track is located in our image. Because the segmentation mask is a polygon, we can check later for irregularities that may indicate obstacles. Focusing on a single class lowers complexity by simplifying both training and deployment. We avoid labeling every possible obstacle class and concentrate on perfecting rail detection. Since we train specifically on rail images, the model learns rail-specific features (e.g., sleepers, ballast transitions) and remains robust under different weather and track conditions.

4.3 Obstacle Recognition

4.3.1 Motivation and Alternatives

Once we know where the rail lies, we need to detect objects that may obstruct it. We examine three main strategies:

- ▻ **Generic Segmentation Model:** We initially test a general-purpose network that attempts to segment every visible object. This approach finds objects outside our training data, but it runs too slowly for real-time usage.

- ▻ **Extended Rail-Segmentation Model:** We consider adding an “obstacle” label to the rail model itself. However, this choice significantly increases manual annotation, model size, and training time.

- ▻ **Two-Stage Pipeline:** We maintain a specialized rail segmentation model and introduce a separate object detection network. We then filter detections based on whether they intersect the rail polygon.

We adopt the two-stage pipeline because it achieves a practical balance between speed and accuracy, as each stage focuses on a narrower task. For obstacle recognition, we employ a real-time detection model (e.g., RT-DETR) that locates bounding boxes around objects and classifies them. We then check whether these bounding boxes overlap with the segmented rail area.

4.3.2 Reasons for Our Choice

A dedicated detection model (e.g., RT-DETR) runs quickly while still accurately finding obstacles. A low inference time allows the pipeline to process the image signal at a high frame rate, resulting in an overall low reaction time that is crucial for providing information about track safety as fast as possible. We can improve or replace the obstacle detector independently of the rail segmentation network, which is important when we receive new training data or when other improvements become available. Moreover, we only need to label obstacles in a generic sense, rather than enumerating all classes within the same network that segments rails. In addition, we can use obstacle datasets from other domains, such as the automotive sector, which increases the variety in training data and results in better obstacle detection accuracy.

4.4 Scoring

scoring is a critical component of our system, as it converts the outputs of rail segmentation and obstacle recognition into a single, comprehensive metric that informs an autonomous controller (in our case, a control system that controls the train and decides autonomously when to brake or accelerate) . A high score indicates that the track presents significant danger. Either because of compromised rail integrity or hazardous obstacles. Whereas a low score signifies that the track appears safe. This unified metric simplifies the control system's decision process, ensuring a prompt and clear response to potential risks.

4.4.1 Purpose and Structure

Our final step merges the results of rail segmentation and obstacle recognition into a single metric that an autonomous controller can interpret easily. We call this metric the *Overall Score*. It comprises two sub-scores:

1. **Rail Score:** Reflects how clearly and continuously we detect the rail in the image. Large gaps or irregularities in the rail polygon may indicate obstructions or poor visibility.

2. **Obstacle Score:** Quantifies the danger level of each detected object based on the distance to the rail, the distance to the train, intersection with the rail polygon, and movement direction.

We compute each sub-score separately and then take the maximum:

$$\text{Overall Score} \;=\; \max\Big(\text{Rail Score},\; \max(\text{Obstacle Scores})\Big). \tag{4.4.1}$$

This design ensures that the final score rises to a critical level if either the rail itself appears compromised or any single obstacle poses a significant threat.

4.4.2 Scoring Details

Rail Thresholds. The thresholds provide the scoring system with information about the safe distance at which the score must increase. Users can adjust these thresholds to suit different camera heights or track conditions. The first threshold sets the expected length of the track visible in the image; in a flat environment, the position of the horizon in the image should be used. Hilly or mountainous environments may require other thresholds due to the camera's viewing angle. Additional thresholds determine a zone for gradual braking and a zone where the train must stop immediately. The threshold values must correspond to the train's braking distance, which varies with the train's speed; therefore, the thresholds should be set dynamically to account for different braking distances.

Rail Score. The scoring system measures the integrity of the rail polygon by examining discontinuities, dents, or notches in its segmented shape. If an irregularity is detected, we use the bottom of the irregularity as the effective length of the rail polygon. If this length falls below a predetermined threshold, the *Rail Score* increases, which signals a potential obstruction.

Obstacle Score. For each detected obstacle, we compute:

▷ **Base Value from Distance:**

$$\text{Base} = \max\big(0,\; 100 - d_{\text{rail}}\big), \tag{4.4.2}$$

where d_{rail} is the obstacle's distance to the rail polygon (in pixels or a comparable unit). Closer obstacles yield higher base values.

▷ **Motion Bonus:**

$$\text{MotionBonus} = \begin{cases} +20, & \text{if the obstacle moves toward the rail,} \\ 0, & \text{if the obstacle is stationary,} \\ -20, & \text{if the obstacle moves away.} \end{cases} \tag{4.4.3}$$

We estimate movement by comparing distances across consecutive frames.

▷ **Intersection Penalty:**
If the obstacle polygon directly intersects the rail, we raise its danger score significantly (often to a maximum) because it occupies the drivable path.

▷ **Train Proximity Bonus:**

$$\text{TrainBonus} = \max\big(0,\ 100 - d_{\text{train}}\big), \tag{4.4.4}$$

where d_{train} is the distance from the obstacle to the train's camera (in pixels or a comparable unit). Obstacles that are closer to the train yield higher bonus values.

We sum these factors to produce an *Obstacle Score*:

$$\text{Obstacle Score} \ = \ \alpha \cdot \text{Base} \ + \ \beta \cdot \text{MotionBonus} \ + \ \gamma \cdot \text{IntersectionFlag} \ + \ \delta \cdot \text{TrainBonus}, \tag{4.4.5}$$

where α, β, γ, and δ are user-tunable weights.

4.4.3 Thresholds and Tunable Variables

We implement several thresholds and weighting parameters to adapt the scoring system to different rail environments:

▷ **Rail Thresholds:** Define the braking zones and the maximum visible track length.

▷ **Distance Threshold:** Sets the point at which an obstacle's proximity becomes critical (e.g., 100 pixels).

▷ **Motion Bonus Factors:** Allow us to calibrate how strongly approaching motion increases the danger score.

▷ **Intersection Weight:** Controls the penalty for obstacles that intersect the rail polygon.

By adjusting these parameters, we can tune the system for short-range yard operations or long-range mainline travel, balancing false positives against missed detections.

4.5 Summary

We build a three-part concept—rail segmentation, obstacle recognition, and scoring—to address the core challenge of identifying obstacles on rail tracks in real time. Rail segmentation locates the safe path, obstacle recognition detects and localizes potential hazards, and the scoring mechanism merges both outputs into a single, actionable metric. This modular design simplifies development, supports robust detection, and enables flexible tuning for different use cases. In the following chapters, we detail how we implement and evaluate each part of this concept.

Chapter 5

Implementation

As described in Chapter 4, we structure our system into three major components: rail segmentation 4.2, obstacle detection 4.3, and scoring 4.4. In this chapter, we explain how we realize that concept in practice, including details on data preparation, model training, and the computation of the final score. Figure 5.1 presents our complete pipeline; Figure 5.2 shows how we deploy modules on an edge device or server. We illustrate our approach to detecting rail irregularities with vertical "rays" in Figure 5.3, and we display a typical annotated output in Figure 5.4.

5.1 Data Preparation and Model Training

For rail segmentation, we use the publicly available RailSem19 dataset. We convert its annotations into a YOLO-compatible format, which ensures consistent and high-quality labels for the rail region without the need for manual annotation. For obstacle detection, we use a pre-trained RT-DETR model for object detection.

Field Test Data Collection. We conduct extensive field tests on the *REAKT* track, a 17 km long route between Malente and Lütjenburg. We collect approximately 60 GB of data and 10 hours of video material using three different cameras. Each video is recorded at 720p resolution and 60 fps. One camera employs a global shutter, while the other two use rolling shutters; notably, one of the rolling shutter cameras lacks an Infrared (IR) filter. Due to distortions in rolling shutter images, we use the global shutter camera for field testing.

Data Augmentation. We perform data augmentation by randomly flipping and cropping images and adjusting brightness levels. These augmentations improve performance under moderate conditions, such as minor variations in illumination or camera angle. However, when video quality is severely compromised due to adverse weather or insufficient lighting, the model fails to perform reliably even with these augmentations.

Model Training. We adopt two distinct model architectures to address our tasks:

- **Rail Segmentation Model:** We train a specialized segmentation network based on the YOLOv11 architecture (model size m) for a single class (rail). By using the converted annotations from the RailSem19 dataset, we eliminate manual labeling and reduce model

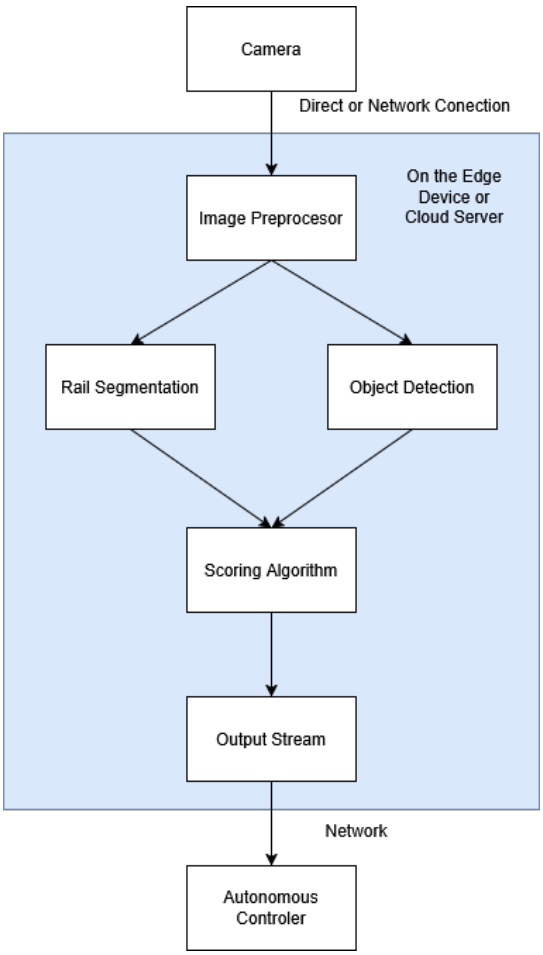

Figure 5.1. Pipeline overview.

complexity, and we include the other classes from the dataset during training to minimize false positive labels. This approach ensures that objects resembling rails but that are not rails receive distinct labels and are not mistakenly classified as rail..

▻ **Obstacle Detection Model:** We deploy one on the COCO dataset pre-trained RT-DETR model (model size l) that leverages a ViT backbone instead of a conventional CNN. The weights are pre-trained on the COCO dataset, ensuring robust detection performance.

During training, we systematically monitor the intersection-over-union (IoU) for rail segmentation and the average precision for obstacle detection. *Intersection over union* measures the overlap between the predicted segmentation mask and the ground truth annotation, computed as the ratio of the area of overlap to the area of union. We stop training when the validation scores converge, thus preventing overfitting and ensuring real-time performance as required by our concept.

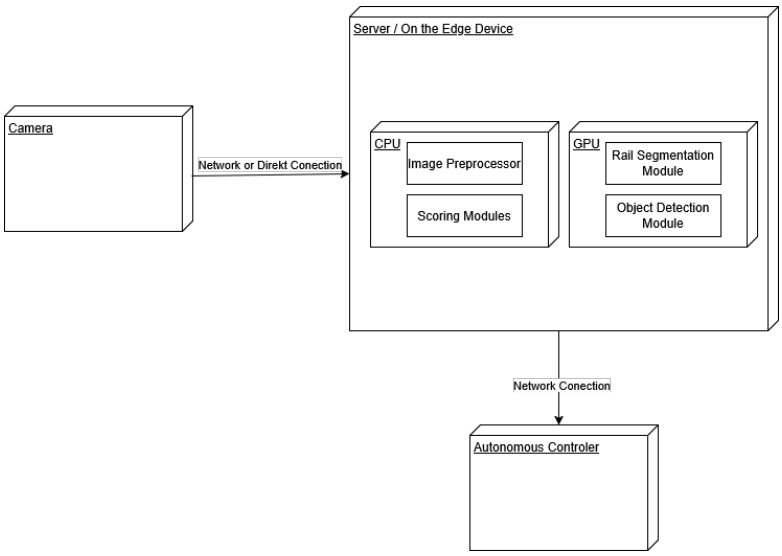

Figure 5.2. Module placement.

5.2 Real-Time Pipeline

We implement a real-time pipeline that continuously processes incoming video frames. Figure 5.1 illustrates the pipeline; it shows how frames are processed by the rail segmentation and obstacle detection modules before being integrated into the scoring mechanism where an output stream is generated. The pipeline operates as follows:

1. **Frame Acquisition:** We capture frames from an Real Time Streaming Protocol (RTSP) stream or a direct camera input, processing only the most recent frame to minimize latency.

2. **Preprocessing:** We resize frames to the input resolution expected by our trained models, preserving the aspect ratio.

3. **Rail Segmentation:** Our model outputs a mask of the rail region. We convert this mask into a *rail polygon* using geometric libraries, which defines the rail region in the image.

4. **Obstacle Detection:** In parallel, we run the pre-trained RT-DETR model to detect and classify objects. We track these objects over time by matching detections across consecutive frames.

5. **Score Calculation:** We compute the *Rail Score* by analyzing the rail polygon for discontinuities and determine the *Obstacle Score* based on object proximity and intersection with the rail region. The *Overall Score* is then generated as the maximum of these scores.

6. **Result Streaming:** We stream the *Overall Score* along with the annotated video frames over a lightweight web interface.

5.3 Scoring and Irregularity Detection

After rail segmentation and obstacle detection, we merge both outputs to compute a final *Overall Score* (see Section 4.4 in the *Concept* chapter). This unified metric enables an autonomous controller to take prompt actions, such as applying brakes or reducing speed.

5.3.1 Rail Score

We compute a *Rail Score* by analyzing the rail polygon for irregularities. As depicted in Figure 5.3, we cast multiple vertical "rays" into the rail mask and measure their uninterrupted lengths. If a rail discontinuity (e.g., a large gap) is detected, the effective rail length decreases, and the *Rail Score* increases accordingly, signaling a potential obstruction or reduced visibility.

Figure 5.3. Detecting irregularities.

5.3.2 Obstacle Score

For each detected object, the *Obstacle Score* considers:

- ▻ Distance to the rail polygon,
- ▻ Intersection with the rail,
- ▻ Motion relative to the train, and
- ▻ Proximity to the camera.

We apply a weighted sum of these factors, following the formulas in Section 4.4 of the *Concept* chapter. For example, if an object moves closer to the rails, we add a positive motion bonus, and if it intersects the polygon, we apply a significant penalty, often raising the score to a maximum.

5.3.3 Overall Score

We compute the *Overall Score* as the maximum of the *Rail Score* and the highest *Obstacle Score*. This method ensures that any critical condition—whether a rail discontinuity or an object on the track—results in an elevated score, prompting immediate action.

5.4 Multi-Threaded Integration

We implement the pipeline using multiple threads to achieve real-time performance. One thread captures frames from the camera, while another executes the inference modules (segmentation and detection) in parallel. We synchronize shared data structures to avoid race conditions, and a final thread streams the annotated results over a lightweight web interface. Figure 5.2 illustrates how these modules can run on a single edge device or a remote server.

5.5 Annotated Output

Figure 5.4 shows a typical frame produced by our system. We color the segmented rail region, draw bounding boxes for detected objects, and display both the *Rail Score* and the *Obstacle Score*. We also overlay the *Overall Score*, which summarizes the track's safety status.

Figure 5.4. Annotated detection.

5.6 Thresholds and Tunable Parameters

A critical aspect of our scoring system is the careful selection of thresholds and tunable variables, which adapt the system to varying operational conditions. In our implementation, using a frame height of 1080 pixels, we chose the following specific values:

▻ **Rail Thresholds:**

- ▻ T1 = $\left(\frac{426}{720}\right) \times$ `frame_height` $\approx 426\,\text{px}$,
- ▻ T2 = $\left(\frac{240}{720}\right) \times$ `frame_height` $\approx 240\,\text{px}$,
- ▻ T3 = $\left(\frac{187}{720}\right) \times$ `frame_height` $\approx 187\,\text{px}$.

T1 represents the expected rail length when the rail is clearly visible up to the horizon. T2 and T3 define intermediate zones for gradual braking and immediate stopping. In real train operations, where the braking distance is longer, T2 and T3 would be adjusted further down in the frame.

- ▻ **Center Offset Tolerance:** Set to 200 px, this parameter helps select the rail polygon that is most centrally located in the image.
- ▻ **Obstacle Scoring Parameters:**
  - ▻ The *base score* is computed as $\max\left(0,\, 100 - d_{\text{rail}}\right)$, where d_{rail} is the distance from the obstacle to the rail polygon.
  - ▻ The *motion bonus* is +20 if an obstacle is moving toward the rail and -20 if moving away; no bonus is added if the obstacle is stationary.
  - ▻ When an obstacle's bounding box intersects the rail polygon, its danger score is forced to at least 80, and if the obstacle is in the lower third of the frame (i.e., $y_{\max} > \frac{2}{3} \times$ `frame_height`) with negligible motion, the score is set to 100.
  - ▻ The *train proximity bonus* is computed as $\max\left(0,\, 100 - d_{\text{train}}\right)$, where d_{train} is the vertical distance from the bottom of the frame to the bottom of the detected object.

These values were determined based on our experimental setup using 720p video. They provide a balance between sensitivity and false alarms, while also allowing for adjustments when transitioning to real train operations with different braking requirements.

5.7 Summary

This chapter details the practical implementation of the system described in Chapter 4. We use the RailSem19 dataset for rail segmentation by converting its annotations to a YOLO-compatible format, and we deploy a pre-trained RT-DETR model for obstacle detection. We train a specialized rail segmentation model using the YOLOv11 architecture (model size m) and employ a pre-trained RT-DETR model (model size l) for object detection. The system integrates both models into a multi-threaded, real-time pipeline that computes an *Overall Score* to indicate potential hazards. The annotated outputs visually convey track continuity, detected objects, and safety status, enabling an autonomous controller to respond promptly to obstructions.

Chapter 6

Evaluation

This chapter provides a comprehensive evaluation of our obstacle detection system. We begin by explaining the key metrics and diagrams, including the confusion matrix, precision-recall curves, F1 curves, and confidence curves. We then discuss how inference speed, measured in Frames Per Second (FPS), directly affects detection latency. Finally, we present our field test results, showing both successful and failure cases, and discuss what these results mean for system performance in real-world conditions.

6.1 Understanding the Metrics and Diagrams

Before presenting the results, it is essential to define the key evaluation metrics:

Precision is defined as the ratio of true positive predictions to the total number of positive predictions. It measures the accuracy of the model's positive detections. **Recall**, on the other hand, is the ratio of true positive predictions to the total number of actual positives, indicating the model's ability to capture all relevant instances. The **F1 score** is the harmonic mean of precision and recall, combining both metrics into a single value that balances the trade-off between false positives and false negatives.

In our evaluations, we use **confidence** to quantify the model's certainty in its predictions. **Confidence** is a score—typically between 0 and 1—that reflects the probability that a given prediction is correct. By adjusting the confidence threshold, we can control which predictions are considered valid; higher thresholds generally reduce false positives while possibly lowering recall, whereas lower thresholds can increase recall at the cost of precision.

6.2 Confusion Matrix Analysis

The confusion matrix compares the ground-truth labels with the model's predictions on a pixel level. In our normalized confusion matrix (Figure 6.1), approximately 87% of the rail pixels are correctly classified. This value exceeds the typical threshold of 85% for robust performance, confirming that the model reliably distinguishes rail pixels from non-rail pixels. A low misclassification rate (below 15%) reinforces our confidence in the model's accuracy.

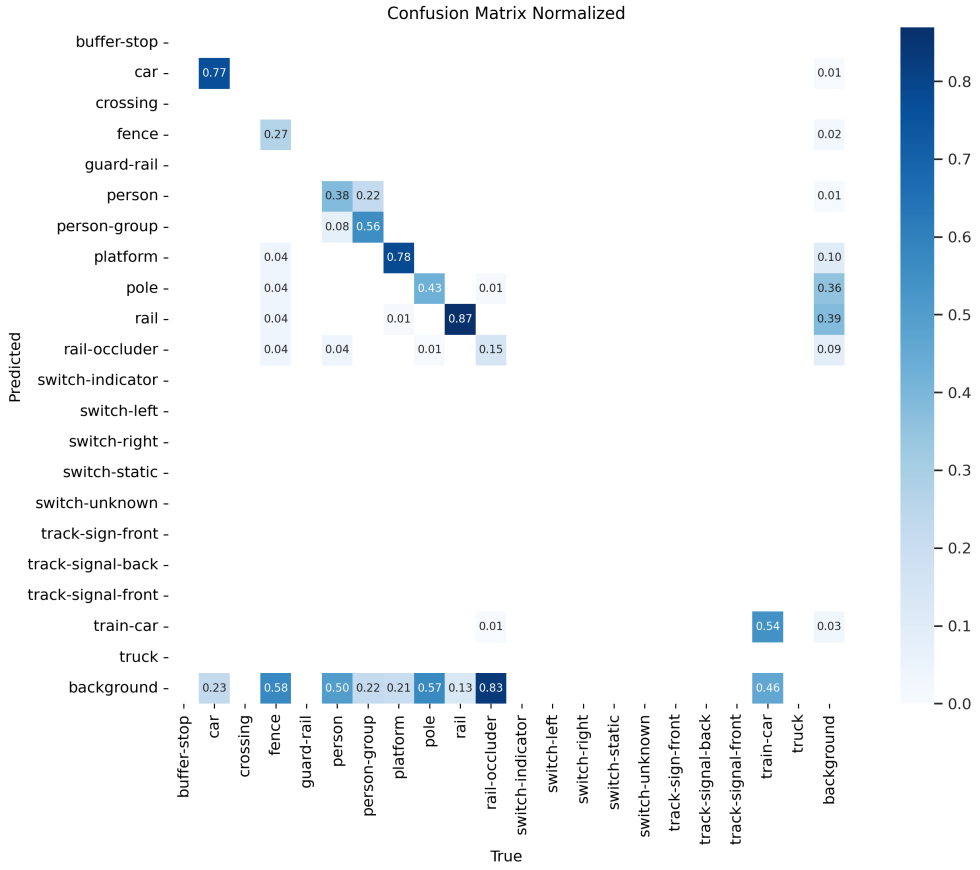

Figure 6.1. Normalized confusion matrix for rail segmentation, showing about 87% correct classification of rail pixels.

6.3 Bounding Box Detection Metrics

The bounding box detection metrics assess how accurately our model localizes the rail region. In Figure 6.2, the precision-recall curve indicates that at a confidence threshold around 0.8 the model achieves a precision of approximately 83% and a recall of nearly 88%. Such values are considered excellent, as precision and recall above 80% are generally desired for safety-critical applications. Figure 6.3 shows that the F1 score for bounding box detection peaks around 84%, confirming a good balance between precision and recall. In Figures 6.4 and 6.5, the variation of precision and recall with different confidence thresholds is presented. For all these diagrams, the magenta curve corresponds to the rail region, which is our main area of interest.

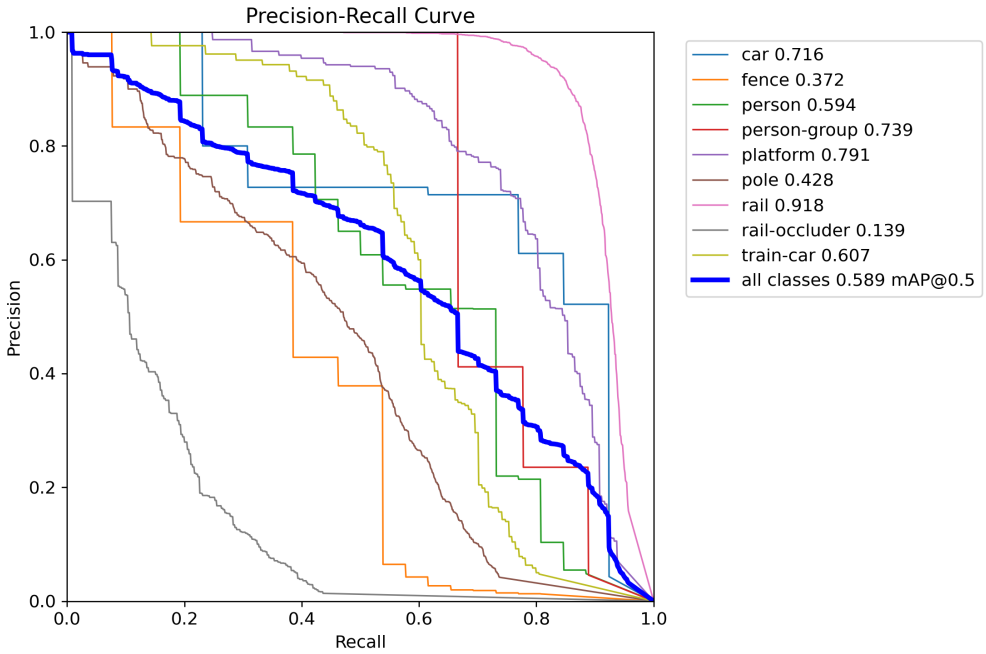

Figure 6.2. Rail bounding box precision-recall curve. At a threshold around 0.8, the magenta curve (rail) shows a precision of about 83% and recall near 88%.

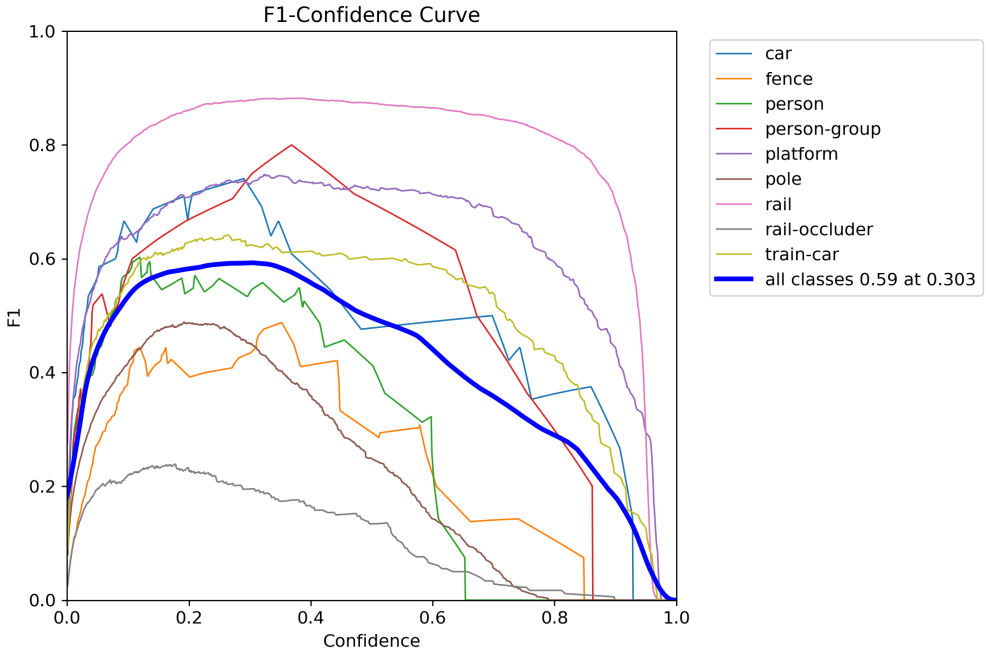

Figure 6.3. Rail bounding box F1 curve, with scores around 84% for the magenta rail curve.

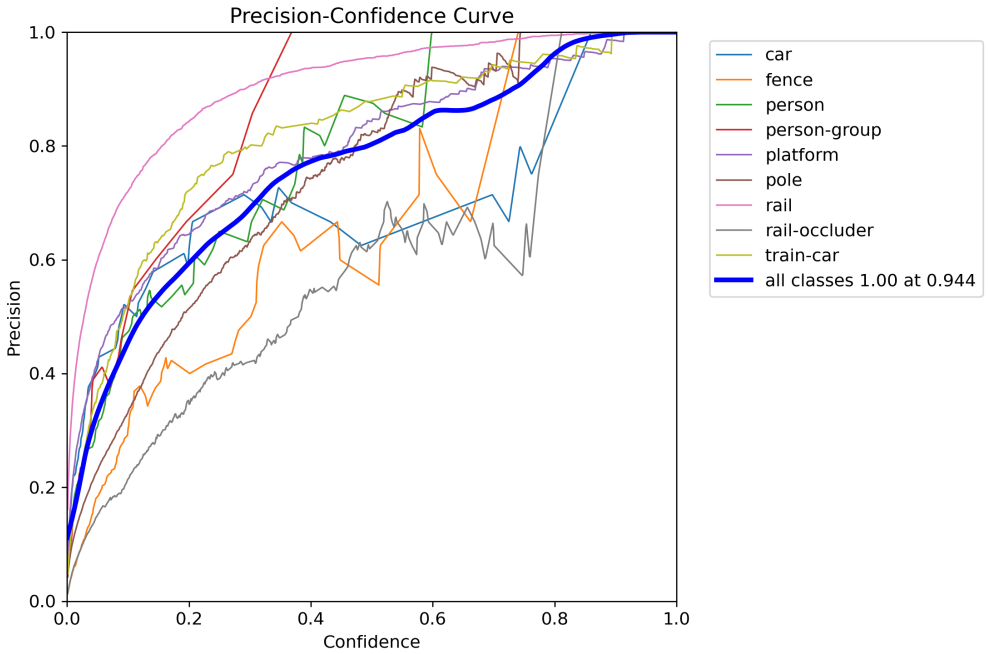

Figure 6.4. Rail bounding box precision curve showing how precision varies with the confidence threshold (magenta: rail).

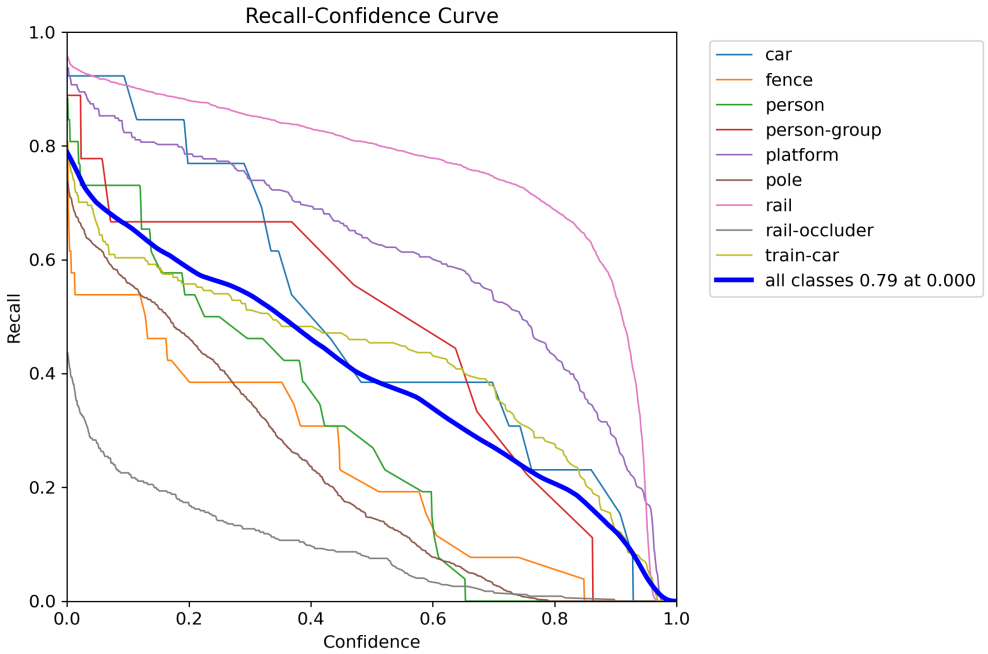

Figure 6.5. Rail bounding box recall curve showing how recall varies with the confidence threshold (magenta: rail).

6.4 Segmentation Mask Metrics

Pixel-level accuracy is assessed through segmentation mask metrics. Figure 6.6 shows the precision-recall curve for the segmentation mask, with precision reaching nearly 95% and recall around 90% for the magenta rail region. This indicates that the model delineates the rail with very little error. Figure 6.7 displays the F1 curve for the segmentation mask, with scores peaking around 80%. Figures 6.8 and 6.9 further illustrate the variation of precision and recall with the confidence threshold. Overall, these high values demonstrate that the model performs excellent segmentation of the rail region.

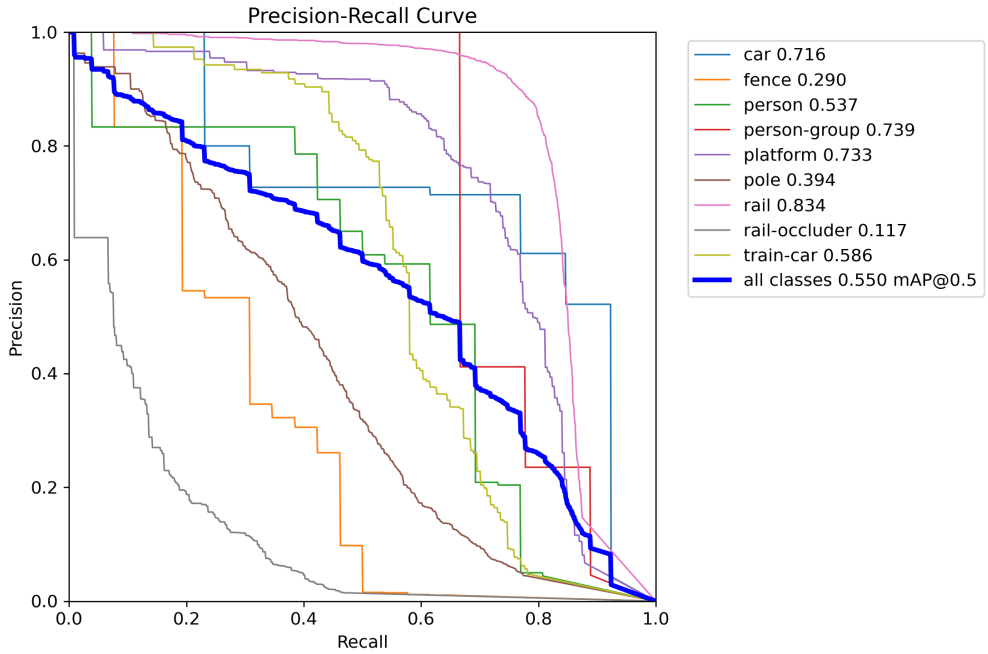

Figure 6.6. Rail segmentation mask precision-recall curve, with precision near 95% and recall around 90% (magenta: rail).

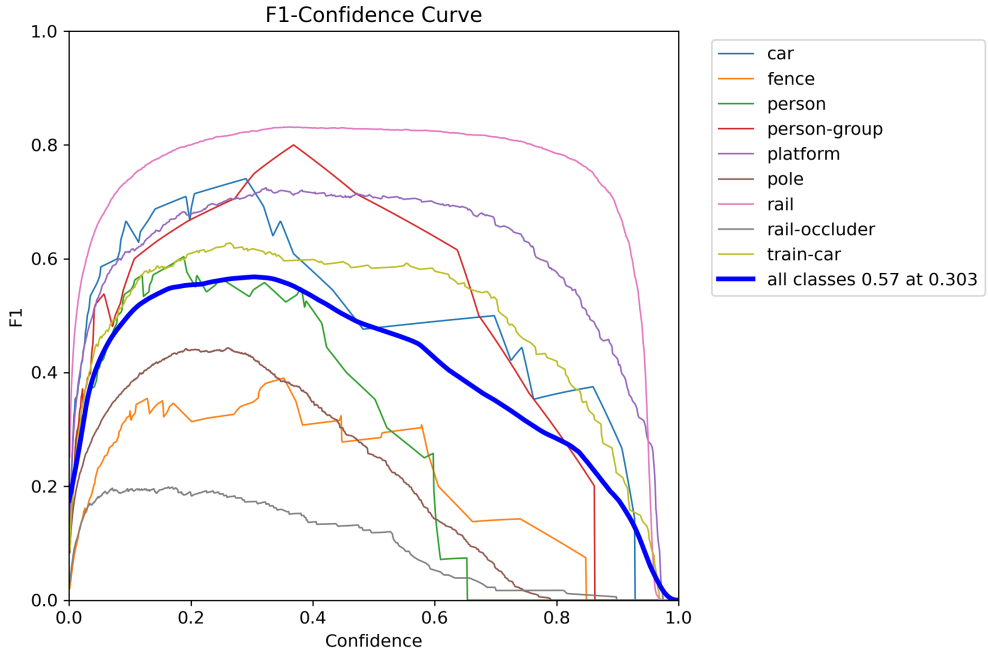

Figure 6.7. Rail segmentation mask F1 curve, with scores around 80% (magenta: rail).

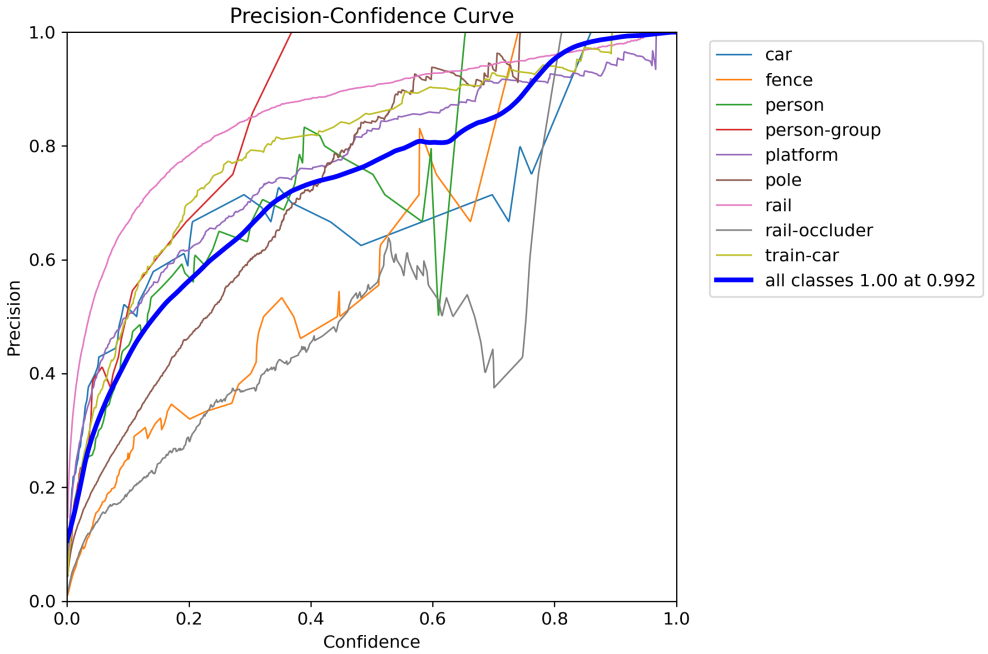

Figure 6.8. Rail segmentation mask precision curve (magenta indicates the rail performance).

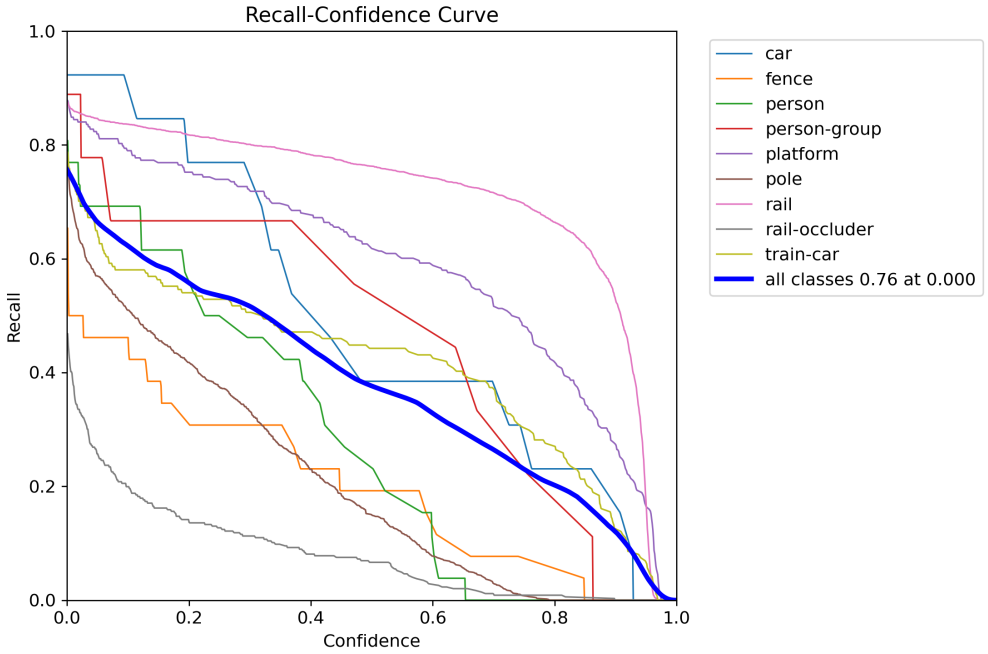

Figure 6.9. Rail segmentation mask recall curve (magenta indicates the rail performance).

6.5 Pipeline Speed

The pipeline speed is critical because it determines how quickly the system processes new frames and, consequently, how fast it can detect obstacles. In our field tests conducted on the REAKT track, our system achieved an average processing speed of approximately 9 FPS. This processing speed translates into a per-frame delay of about 111 ms, which directly affects the detection latency. A higher FPS—for example, 15–20 FPS (corresponding to 67–50 ms per frame)—would reduce the latency further. Considering that the typical human reaction time to visual stimuli is around 200–250 ms, our current performance is on the under edge of realtime capable, but improvements in FPS could enhance system safety and responsiveness.

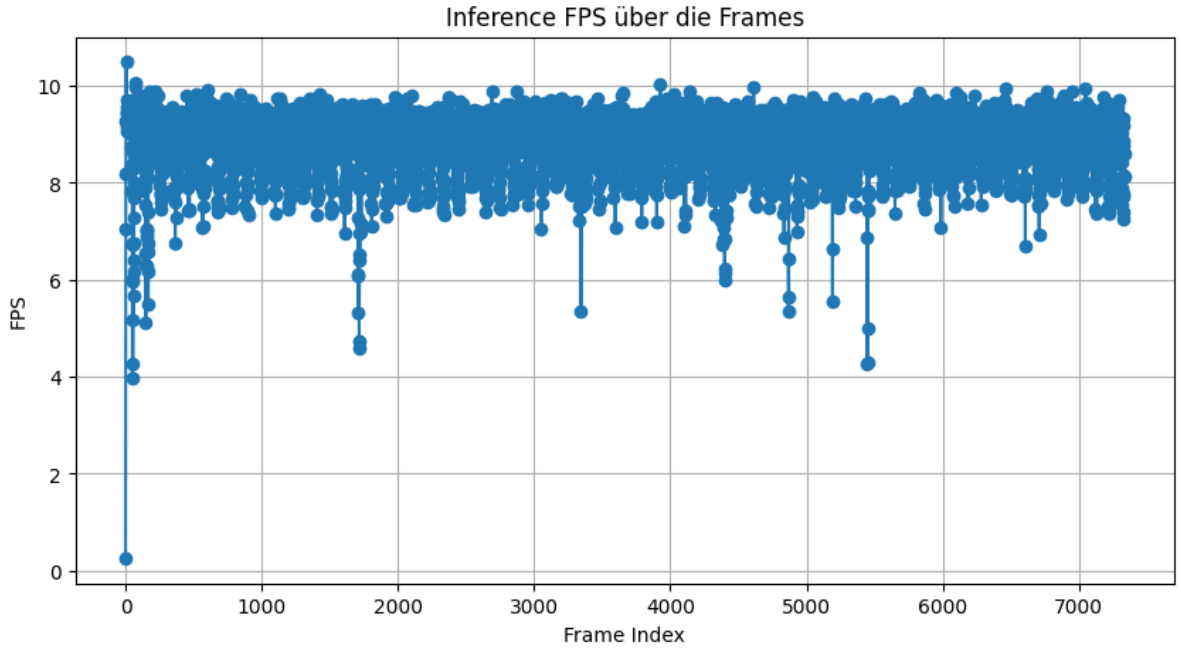

Figure 6.10. Inference speed measured in FPS. Our system processes about 9 FPS, corresponding to a per-frame delay of approximately 111 ms.

6.6 Field Test Results

In addition to controlled experiments, we conducted extensive field tests on the REAKT track. These tests provided valuable insights into real-world performance. In one field test, illustrated in Figure 6.11, the system successfully detected the rail and obstacles, achieving a rail score of approximately 43 and an obstacle score of 80. This image, captured under favorable conditions with high image quality, confirms that the system performs well when the environment is optimal. Conversely, Figure 6.12 shows a field test case where adverse conditions, such as vibration-induced blur and low contrast, resulted in a poor detection

performance with an overall score of around 20. These two images underscore the strengths and limitations of our system in real-world scenarios, highlighting the need for further improvements in image stabilization and sensor fusion to ensure robust operation under all conditions.

Figure 6.11. Field Test: Successful detection example. The system accurately identifies the rail marked green and obstacles marked blue, and orange, with a rail score of approximately 43 and an obstacle score of 80.

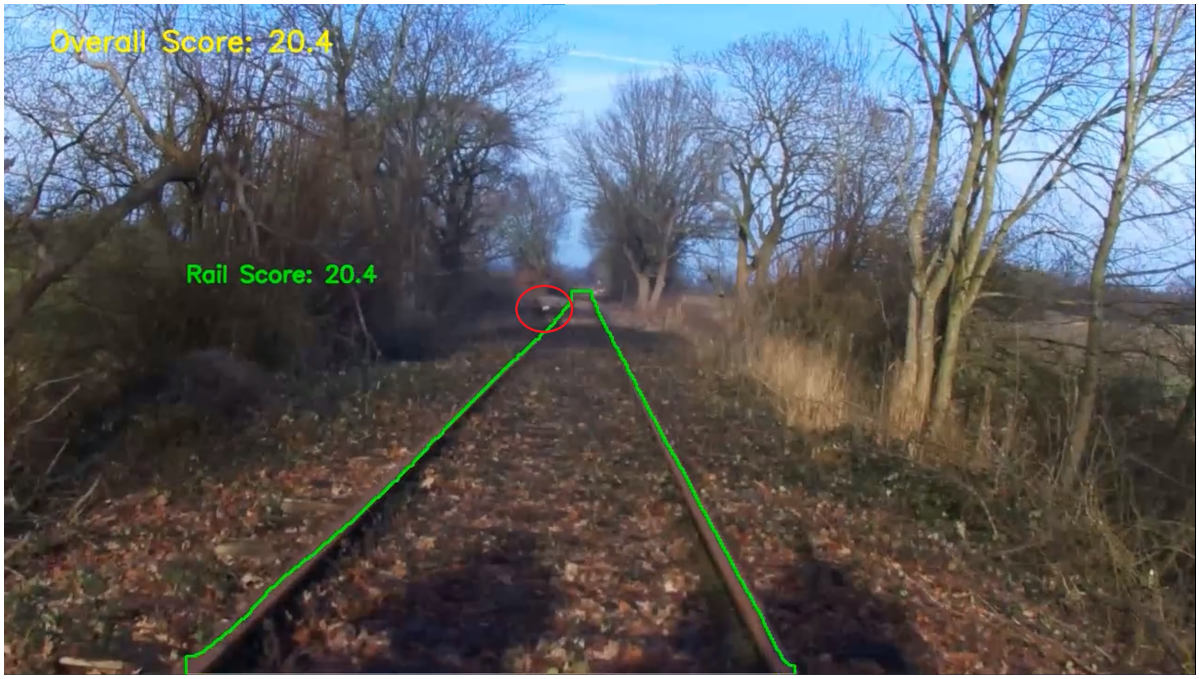

Figure 6.12. Field Test: Detection failure case. Poor image quality due to vibrations leads to an overall score of about 20, indicating missed detections under challenging conditions.

6.7 Discussion and Implications

The evaluation demonstrates that our two-stage approach, which combines dedicated rail segmentation with real-time obstacle detection, meets robust performance criteria. The normalized confusion matrix indicates that about 87% of rail pixels are correctly classified—well above the 85% minimum threshold. The bounding box metrics (with precision around 83%, recall near 88%, and F1 scores around 84%) and the segmentation mask metrics (with precision near 95%, recall around 90%, and F1 scores around 80%) confirm excellent localization and segmentation performance for the rail.

Inference speed is another crucial factor. Our system processes at about 9 FPS, resulting in a per-frame delay of 111 ms. While this is competitive relative to human reaction times (200–250 ms), increasing the FPS to 15–20 would reduce detection latency further, enabling faster system responses and improved safety in dynamic environments. The field test results illustrate that, under real-world conditions, the system performs reliably when conditions are favorable, but its performance degrades in adverse situations—indicating areas for future improvement.

6.8 Summary

In summary, our evaluation confirms that the proposed obstacle detection system achieves robust performance across all key metrics. The normalized confusion matrix, bounding box and segmentation mask curves, and training/validation analysis all indicate performance above the minimum acceptable thresholds. The magenta curves, which represent the rail, are of primary importance in our evaluation. Although the current inference speed of 9 FPS results in a per-frame delay of about 111 ms, which is competitive with human reaction times, further improvements in FPS could enhance system responsiveness, thereby reducing detection latency and increasing overall safety.

Chapter 7

Conclusion

This chapter summarizes the key parts of this thesis, evaluates the current performance of the system, and outlines the necessary steps to achieve production readiness.

7.1 System Capabilities and Production Readiness

We design and implement a dual-scoring, camera-based AI system that supports autonomous rail operation. The system uses a YOLO-based segmentation model to extract rail regions and an RT-DETR detector to identify obstacles. It computes two scores—a Rail Score that evaluates the integrity of the rail segmentation, and an Obstacle Score that measures obstacle proximity, motion, and intersection with the rail. By combining these scores into an Overall Score, the system provides a clear, actionable safety metric.

Our experiments demonstrate that the system works reliably in controlled environments and performs real-time obstacle detection when conditions are favorable. It accurately identifies rail regions and detects obstacles, enabling prompt decision-making for autonomous control. However, the system encounters several limitations. For instance, if the train's braking distance exceeds the vision range, the system cannot guarantee safe braking. The system struggles with adverse weather, low-light conditions, high-vibration scenarios, and blurred imagery, which all reduce detection accuracy. Moreover, the current inference speed and reliance solely on camera data limit its robustness compared to multi-sensor approaches.

To prepare the system for production, additional sensor modalities—such as LiDAR, radar, and thermal imaging—should be integrated to supplement visual data and improve detection under varied conditions. The inference pipeline should be optimized to reduce latency and increase frame rates, enhancing of the image stabilization also is necessary. Data augmentation, and multi-object tracking to handle dynamic environments would be good to get ready for production. Extensive field tests under diverse operational conditions, along with hardware optimizations, are essential to meet the stringent safety and reliability standards required for commercial autonomous rail applications.

7.2 Future Work

Future work should expand the current conceptual framework by exploring enhancements aimed at advancing autonomous rail operation. One promising direction is the integration of additional sensor modalities. By incorporating sensors such as LiDAR, thermal imaging,

and radar, the system benefits from a richer set of environmental data. This sensor fusion approach provides complementary perspectives, enabling more robust detection and improved segmentation performance even in challenging conditions, such as poor lighting or adverse weather.

In addition to enhancing sensor input, further research should focus on developing and integrating advanced multi-object tracking algorithms. Improved tracking facilitates a more precise estimation of object motion and enhances the consistency of obstacle monitoring over time. Such enhancements are crucial for accurately differentiating between dynamic and static objects along the rail, which in turn supports better decision-making in an autonomous driving context.

Another key area for future investigation is extending the segmentation model to handle multiple classes directly. By enabling the model to delineate the rail and recognize common obstacles in a single stage, the overall detection pipeline becomes significantly simpler. This multi-class segmentation approach has the potential to reduce computational complexity and streamline the processing workflow, making the system more efficient and easier to deploy.

Moreover, conducting extensive field tests across diverse operational scenarios is essential. These real-world evaluations allow researchers to validate the conceptual approach and fine-tune scoring thresholds under various conditions, such as differing weather patterns, lighting situations, and dynamic track environments. Such comprehensive testing is indispensable for ensuring that the system adapts effectively to practical applications in autonomous rail driving.

Finally, hardware optimization remains an important aspect for future development. Investigating accelerated implementations and exploring faster model versions are necessary to achieve higher frame rates on embedded devices. This optimization not only enhances the real-time performance of the system but also paves the way for its integration into commercial autonomous rail solutions.

Bibliography

[Bea24] Beamagine2024. *L3cam: advanced sensor fusion for autonomous railway systems*. https://www.beamagine.com/l3cam. Accessed: 2025-03-13. 2024.

[Bra00] Gary R. Bradski. "The opencv library". In: *Dr. Dobb's Journal of Software Tools* 25.11 (2000), pp. 120, 122–125. URL: http://www.drdobbs.com/open-source/the-opencv-library/184404319.

[DBK+21] Alexey Dosovitskiy et al. "An image is worth 16x16 words: transformers for image recognition at scale". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2021. DOI: 10.48550/arXiv.2010.11929. URL: https://arxiv.org/abs/2010.11929.

[EYY24] EYYES2024. *Raileye3d: ai-powered 3d sensing for railway safety*. https://www.eyyes.com/raileyes3d. Accessed: 2025-03-13. 2024.

[Fut24] Futurail2024. *Futurail: ai-driven autonomous train operation*. https://www.futurail.ai. Accessed: 2025-03-13. 2024.

[GVP+20] Julian Gleichauf, Julian Vollet, Carsten Pfitzner, Peter Koch, and Stefan May. "Sensor Fusion Approach for an Autonomous Shunting Locomotive". In: *Informatics in Control, Automation and Robotics*. Springer, Cham, 2020, pp. 603–624. DOI: 10.1007/978-3-030-40832-3_30.

[JQ24] Glenn Jocher and Jing Qiu. *Ultralytics yolo11*. Version 11.0.0. 2024. URL: https://github.com/ultralytics/ultralytics.

[LBB+98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and C. Lawrence Zitnick. "Microsoft COCO: common objects in context". In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[MTM12] A. Mogelmose, M.M. Trivedi, and T.B. Moeslund. "Vision-based obstacle detection for autonomous vehicles: a survey". In: *IEEE Transactions on Intelligent Transportation Systems* 13.3 (2012), pp. 988–1000.

[RAH+22] Fahim Ur Rahman, Md. Tanvir Ahmed, Md. Mehedi Hasan, and Nusrat Jahan. "Real-time obstacle detection over railway track using deep neural networks". In: *Procedia Computer Science* 215 (2022). 4th International Conference on Innovative Data Communication Technology and Application, pp. 289–298. ISSN: 1877-0509.

DOI: https://doi.org/10.1016/j.procs.2022.12.031. URL: https://www.sciencedirect.com/science/article/pii/S1877050922021020.

[RDG+16] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. "You only look once: unified, real-time object detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[RFM21] Danijela Ristić-Durrant, Marten Franke, and Kai Michels. "A Review of Vision-Based On-Board Obstacle Detection and Distance Estimation in Railways". In: *Sensors* 21.10 (2021), p. 3452. DOI: 10.3390/s21103452.

[Sie24] Siemens Mobility. *Innovative obstacle detection system is being tested for the first time on the Berlin S-Bahn*. Press Release. July 2024. URL: https://press.siemens.com/global/en/pressrelease/innovative-obstacle-detection-system-being-tested-first-time-berlin-s-bahn.

[TKT+23] Rustam Tagiew, Pavel Klasek, Roman Tilly, Martin Köppel, Patrick Denzler, Philipp Neumaier, Tobias Klockau, Martin Boekhoff, and Karsten Schwalbe. "Osdar23: open sensor data for rail 2023". In: *2023 8th International Conference on Robotics and Automation Engineering (ICRAE)*. IEEE, Nov. 2023, pp. 270–276. DOI: 10.1109/icrae59816.2023.10458449. URL: http://dx.doi.org/10.1109/ICRAE59816.2023.10458449.

[YWS+18] Tao Ye, Baocheng Wang, Ping Song, and Juan Li. "Automatic Railway Traffic Object Detection System Using Feature Fusion Refine Neural Network under Shunting Mode". In: *Sensors* 18.6 (2018), p. 1916. DOI: 10.3390/s18061916.

[YWS+20] Tao Ye, Baocheng Wang, Ping Song, and Juan Li. "Railway Traffic Object Detection Using Differential Feature Fusion Convolution Neural Network". In: *IEEE Transactions on Intelligent Transportation Systems* (2020). DOI: 10.1109/TITS.2020.2969993.

[ZLX+24] Yian Zhao, Wenyu Lv, Shangliang Xu, Jinman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. "DETRs beat YOLOs on real-time object detection (rt-detr)". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. to appear. 2024. DOI: 10.48550/arXiv.2304.08069.

[ZMZ+19] Oliver Zendel, Markus Murschitz, Marcel Zeilinger, Daniel Steininger, Sara Abbasi, and Csaba Beleznai. "Railsem19: a dataset for semantic rail scene understanding". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2019), pp. 1221–1229. URL: https://www.semanticscholar.org/paper/RailSem19%3A-A-Dataset-for-Semantic-Rail-Scene-Zendel-Murschitz/d32db8a5bf047d34b2ef76a8d6bb62