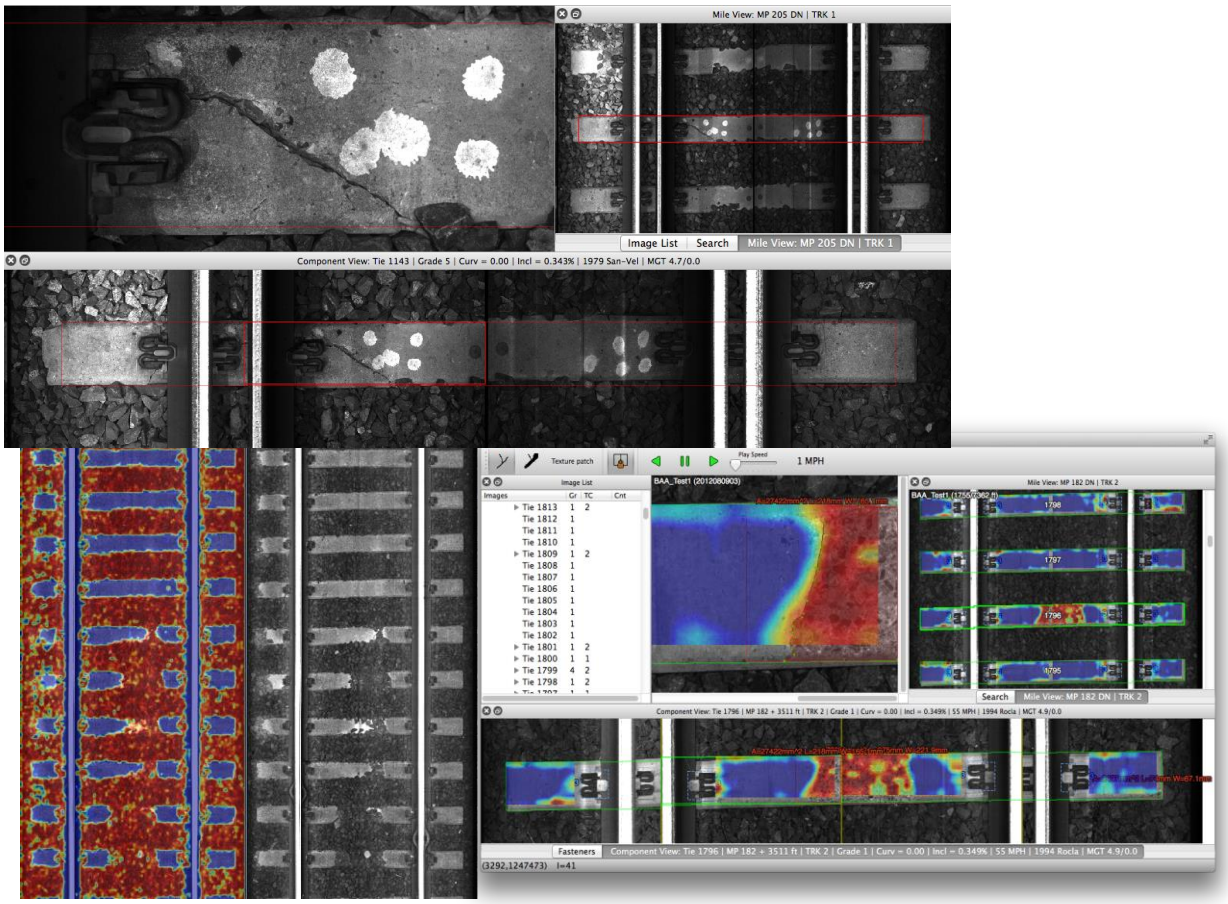
U.S. Department of Transportation

Federal Railroad Administration

Robust Anomaly Detection for Vision-Based Inspection of Railway Components

Office of Research,
Development,
and Technology
Washington, DC 20590



DOT/FRA/ORD-15/23

Final Report
June 2015

NOTICE

This document is disseminated under the sponsorship of the Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof. Any opinions, findings and conclusions, or recommendations expressed in this material do not necessarily reflect the views or policies of the United States Government, nor does mention of trade names, commercial products, or organizations imply endorsement by the United States Government. The United States Government assumes no liability for the content or use of the material contained in this document.

NOTICE

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the objective of this report.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2015	Technical Report – March 2015

4. TITLE AND SUBTITLE
Robust Anomaly Detection for Vision-Based Inspection of Railway Components

5. FUNDING NUMBERS
DOTFR53-13-C-00032

6. AUTHOR(S)
Rama Chellappa, Xavier Gibert, Vishal Patel

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Maryland
A. V. Williams Building
College Park, MD 20742-3275

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Department of Transportation
Federal Railroad Administration
Office of Railroad Policy and Development
Office of Research and Development
Washington, DC 20590

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
DOT/FRA/ORD-15/23

11. SUPPLEMENTARY NOTES
COR: Leith Al-Nazer, Cameron Stuart

12a. DISTRIBUTION/AVAILABILITY STATEMENT
This document is available to the public through the FRA Web site at http://www.fra.dot.gov.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
The Computer Vision Laboratory at the University of Maryland has completed a two-year research project in which it developed a library of tools and algorithms that are used to inspect railway tracks with machine vision technology. This technology has been integrated into a distributed computing framework and a user-friendly review package with a client-server interface. This framework is built on top of high-quality open-source C++ software libraries and can run on many different platforms, including Windows, Linux, OS X, and iOS. The software has been thoroughly tested and ENSCO, Inc. has successfully used these tools during the FRA-sponsored project that evaluated the degradation rates of concrete ties on Amtrak's Northeast Corridor. Our final deliverable contains algorithms for crack detection, fastener detection and classification, as well as semantic segmentation for material classification and anomaly detection.

14. SUBJECT TERMS
visual track inspection, computer vision, concrete tie conditions, fastener inspection, tie inspection, crack detection, deep learning, distributed computing

15. NUMBER OF PAGES
58

16. PRICE CODE

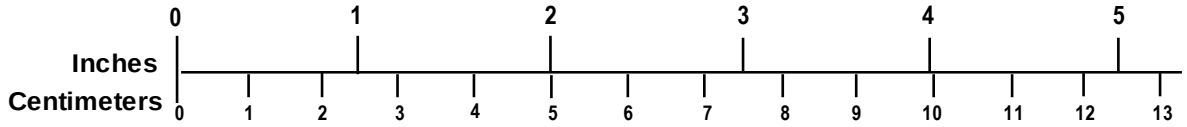
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

NSN 7540-01-280-5500
Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

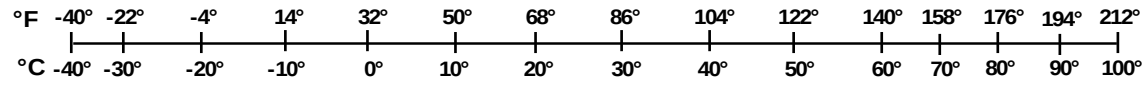
METRIC/ENGLISH CONVERSION FACTORS

ENGLISH TO METRIC	METRIC TO ENGLISH
LENGTH (APPROXIMATE)	**LENGTH (APPROXIMATE)**
1 inch (in) = 2.5 centimeters (cm)	1 millimeter (mm) = 0.04 inch (in)
1 foot (ft) = 30 centimeters (cm)	1 centimeter (cm) = 0.4 inch (in)
1 yard (yd) = 0.9 meter (m)	1 meter (m) = 3.3 feet (ft)
1 mile (mi) = 1.6 kilometers (km)	1 meter (m) = 1.1 yards (yd)
	1 kilometer (km) = 0.6 mile (mi)
AREA (APPROXIMATE)	**AREA (APPROXIMATE)**
1 square inch (sq in, in^2) = 6.5 square centimeters (cm^2)	1 square centimeter (cm^2) = 0.16 square inch (sq in, in^2)
1 square foot (sq ft, ft^2) = 0.09 square meter (m^2)	1 square meter (m^2) = 1.2 square yards (sq yd, yd^2)
1 square yard (sq yd, yd^2) = 0.8 square meter (m^2)	1 square kilometer (km^2) = 0.4 square mile (sq mi, mi^2)
1 square mile (sq mi, mi^2) = 2.6 square kilometers (km^2)	10,000 square meters (m^2) = 1 hectare (ha) = 2.5 acres
1 acre = 0.4 hectare (he) = 4,000 square meters (m^2)	
MASS - WEIGHT (APPROXIMATE)	**MASS - WEIGHT (APPROXIMATE)**
1 ounce (oz) = 28 grams (gm)	1 gram (gm) = 0.036 ounce (oz)
1 pound (lb) = 0.45 kilogram (kg)	1 kilogram (kg) = 2.2 pounds (lb)
1 short ton = 2,000 pounds (lb) = 0.9 tonne (t)	1 tonne (t) = 1,000 kilograms (kg) = 1.1 short tons
VOLUME (APPROXIMATE)	**VOLUME (APPROXIMATE)**
1 teaspoon (tsp) = 5 milliliters (ml)	1 milliliter (ml) = 0.03 fluid ounce (fl oz)
1 tablespoon (tbsp) = 15 milliliters (ml)	1 liter (l) = 2.1 pints (pt)
1 fluid ounce (fl oz) = 30 milliliters (ml)	1 liter (l) = 1.06 quarts (qt)
1 cup (c) = 0.24 liter (l)	1 liter (l) = 0.26 gallon (gal)
1 pint (pt) = 0.47 liter (l)	
1 quart (qt) = 0.96 liter (l)	
1 gallon (gal) = 3.8 liters (l)	
1 cubic foot (cu ft, ft^3) = 0.03 cubic meter (m^3)	1 cubic meter (m^3) = 36 cubic feet (cu ft, ft^3)
1 cubic yard (cu yd, yd^3) = 0.76 cubic meter (m^3)	1 cubic meter (m^3) = 1.3 cubic yards (cu yd, yd^3)
TEMPERATURE (EXACT)	**TEMPERATURE (EXACT)**
[(x-32)(5/9)] °F = y °C	[(9/5) y + 32] °C = x °F

QUICK INCH - CENTIMETER LENGTH CONVERSION



QUICK FAHRENHEIT - CELSIUS TEMPERATURE CONVERSION



For more exact and or other conversion factors, see NIST Miscellaneous Publication 286, Units of Weights and Measures. Price $2.50 SD Catalog No. C13 10286

Updated 6/17/98

Acknowledgements

This work has been made possible through cooperation from the Federal Railroad Administration's Office of Research and Development, the National Railroad Passenger Corporation (Amtrak) and ENSCO, Inc. The authors are grateful to Leith Al-Nazer, FRA's Technical Representative, for his guidance and valuable comments during this project. The authors sincerely thank Felipe Arrate and Daniel Bogachek at the University of Maryland for their invaluable contributions to this project. The authors also thank Boris Nejikovsky at ENSCO for his encouragement to pursue research on this topic, as well as Eric Sherrock for helping with the preparation of the proposal that resulted in this project as well as coordinating this effort. The authors are also grateful to Jeff Henderson and Cindy Scott at ENSCO, as well as their team of track inspectors and image reviewers, for their support with data collection and annotation. Thanks are also extended to Michael Trosino, Michael Craft, Joe Smak and Joe Mascara from Amtrak for granting permission to use the images that made this work possible as well as providing guidance during the project.

Contents

Illustrations

Tables

Executive Summary

This report discusses the design, development and evaluation of a prototype software package for detecting railway track anomalies using computer vision. Although the software's algorithms can be used on different kinds of tracks, the development team has focused on concrete ties in use on high-speed rail (HSR) corridors. Detecting HSR track anomalies is more challenging than finding those on conventional tracks. For example, HSR track requires more frequent inspections and it usually has shorter maintenance windows than conventional track. Recently, the railroad industry has been adopting machine vision technology as a complement to inspecting track manually or other forms of inspection. However, limitations in machine vision system capabilities, including high false positive rates, inhibit the widespread use of these technologies.

The project's objective was to advance the state of the art in automatic anomaly detection for railway track inspection (and other outdoor environments). To accomplish this objective, we have performed basic and applied computer vision research, designed several anomaly detection algorithms, and then implemented a prototype software system that can be programmed to detect cracks, missing/broken fasteners, chips, and crumbling on concrete ties. Because railroads have a need for deployable systems, project development was focused on methods that can be scaled up to the image acquisition data rates that are currently used.

To test the anomaly detection algorithms, the team used 329 miles of concrete tie images that had been collected from three surveys conducted b y ENSCO, Inc. between 2012 and 2013 on Amtrak's Northeast Corridor (NEC). These images were scanned into the Euclid computer cluster at the University of Maryland, then highly customized image annotation tools were created for ENSCO reviewers to generate the ground truth data that was used to evaluate the algorithms. The user interface is a client application that was carefully designed to facilitate evaluation tasks and allow railroad users to quickly review the results of automated detections. This interface connects to a database through an HTTPS interface to generate annotation reviews and detection results. Multiple users can access the database to review the same or different sections of the track.

Three computer vision algorithms are described by this report: 1) A crack detector based on decomposing images into edge and texture components, 2) a missing/broken fastener algorithm based on computer vision features and a statistical classifier, and 3) a crumbling/chipped tie detector based on a material classifier that uses a deep convolutional neural network architecture.

- The crack detector accurately detected the outline of cracks with different sizes and orientations under a variety of background textures, but the ability to accurately differentiate between cracks and other edges in the image is not yet mature enough for practical use.
- The fastener detection algorithm finds the location and type of each fastener, and can determine whether a fastener is broken or missing with a probability of detection of 98% and a false alarm rate of 123 FP/mile (false positives per mile assuming 2.5K ties per mile). However, most false alarms are due to special track work, ballast covering the fastener and other occluding elements. Among clear ties on regular track, the detection rate is 98.36% and the false alarm rate is 38 FP/mile. Among fasteners in good condition,

the algorithm can classify the type of fastener among five categories (PR clip, e clip, fastclip 1, fastclip 2, c-clip, and j-clip) with an accuracy of 98.2%.

- The algorithm for detecting crumbling and chipped ties has two steps. First, the algorithm employs a multiclass detector that has been trained on ten different types of material (ballast, wood, rough concrete, medium concrete, smooth concrete, crumbling concrete, chipped concrete, lubricator, rail, and fastener) to scan each region of the image. This detector uses a deep convolutional neural network to achieve 93.55 percent accuracy. Second, the algorithm estimates the likelihood that the area of the tie affected by crumbling or chipping exceeds a predefined threshold. For defects that are bigger than the 10% threshold (at a false positive rate of 10 FP/mile), the detection rates are 86.06% for crumbling and 92.11% for chips.

To be a fully automated solution that the railroad industry can adopt, more research will be needed to address the following:

- The false alarm rate could be further reduced by developing adaptive algorithms that can operating conditions and the probability distribution of the background clutter.
- The segmentation algorithm used to detect crumbling and chipped ties should be adapted to filter ballast and other obstructing tie elements that currently cause false crack detections
- The dataset used in this report contains only concrete ties from the NEC. In order to validate the performance on more general conditions, more data collection and analysis is required.
- Other potential research areas that include detection and assessment of other track components; learning from poorly labeled data; matching, alignment and change detection of track components; and automated tie grading. Also, the addition to other channels such as depth or color are worth exploring as well.

In conclusion, this report describes a new approach for inspecting railway tracks using recent advances in the area of computer-assissted vision and pattern recognition. The algorithms described in this report have been packaged into an integrated software suite that will allow different railroad users to configure it for their specific needs. We hope that this work will jump-start the research and development of new technological solutions in visual track inspection.

1. Introduction

In this section, the team describes the problems that occur when computer vision technology is used to detect anomalies in railway track components, discusses the modular architecture that was used to design and integrate the software package system, and explains how potential algorithms were evaluated.

1.1 Background

To ensure railroad safety, the condition of railway components must be continuously monitored, Amtrak has discovered that concrete ties encounter different degradation-related problems than wood ties (Smak, 2012). Although concrete ties have a life expectancy of up to 50 years, they may fail prematurely for many reasons:

- Alkali-silicone reaction (ASR), which is a chemical reaction between cement alkalis and non-crystalline (amorphous) silica that forms alkali-silica gel at the aggregate surface (Shehata & Thomas, 2000). These reaction rims have a very strong affinity with water and have a tendency to swell. These compounds can produce internal pressures that are strong enough to create cracks, allowing moisture to penetrate, and thus accelerating the rate of deterioration.
- Delayed Ettringite Formation (DEF) is a type of internal sulfate attack that occurs in concrete that has been cured at excessively high temperatures (Sahu & Thaulow, 2004).
- In addition to ASR and DEF, ties can also develop fatigue cracks due to normal traffic or by being impacted by flying debris or track maintenance machinery. Once small cracks develop, repeated cycles of freezing and thawing will eventually lead to bigger defects.

Fasteners maintain gage by keeping both rails firmly attached to the crossties. According to the Federal Railroad Administration (FRA) safety database[1], in 2013, out of 651 track-related derailments, 27 of them were attributed to gage widening caused by defective spikes or rail fasteners, and another 2 to defective or missing spikes or rail fasteners.

Also, in the United States, regulations enforced by the FRA[2] prescribe visual inspection of high-speed rail tracks with a frequency of once or twice per week, depending on the class of track (which specifies maximum authorized speeds for both freight and passenger trains). These manual inspections are currently performed by railroad personnel, either by walking on the tracks or by riding a hi-rail vehicle at very low speeds. However, such conventional visual inspections of mainlines are subjective and do not produce an auditable visual record. In addition, railroads usually perform automated track inspections with specialized track geometry measurement vehicles within an interval of 30 days or less between inspections. These automated inspections can directly detect gage widening conditions. However, it is preferable to find fastening problems before they develop into gage widening conditions.

Since the pioneering work by Cunningham, Shaw, & Trosino (2000) and Trosino, Cunningham, & Shaw (2002), machine vision technology has been gradually adopted by the railway industry

[1] http://safetydata.fra.dot.gov

[2] 49 CFR 213 -- Track Safety Standards

for track inspection. Their first generation systems could collect images of the railway right of way and store them for later review, but the images were not used in for automated detection of anomalies or defects. As faster processing hardware became available, several vendors began to introduce vision-detection systems with automation capabilities.

The VISyR system, which detects hexagonal-headed bolts using two 3-layer neural networks (NN) running in parallel, is described in Marino, Distante, Mazzeo, & Stella (2007) and De Ruvo, Distante, Stella, & Marino (2009). Both NNs use the 2-level discrete wavelet transform (DWT) of a 24×100 pixel sliding window (their images use non-square pixels) as a input for generating a binary output that indicates the presence of a fastener. However, the first NN uses Daubechies wavelets, while the second uses Haar wavelets; the wavelet decomposition is equivalent to performing edge detection at different scales with two different filters. Both neural networks are trained with the same examples. The final decision is made using the maximum output of each neural network.

The VisiRail system for joint bar inspection is discussed in Gibert, Berry, Diaz, Jordan, Nejikovsky, & Tajaddini (2007) and Berry, Nejikovsky, Gibert, & Tajaddini (2008). The system is capable of collecting images on each rail side, and finding cracks on joint bars using edge detection and a Support Vector Machine (SVM) classifier that analyzes visual features extracted from these edges.

Babenko (2009) describes a fastener detection method based on a convolutional filter bank that is applied directly to intensity images. Each type of fastener has a single filter associated with it, whose coefficients are calculated using an illumination-normalized version of the Optimal Tradeoff Maximum Average Correlation Height (OT-MACH) filter as seen in Mahalanobis, Kumar, Song, Sims, & Epperson (1994). This approach allowed accurate fastener detection and localization and it achieved over 90% fastener detection rate on a dataset of 2,436 images. However, the detector was not tested on longer sections of track.

Resendiz, Hart, & Ahuja (2013) discusses how the authors classified textures with a bank of Gabor filters then used an SVM to determine the location of rail components such as crossties and turnouts. They also use the MUSIC algorithm to find spectral signatures to determine expected component locations. In Li, Trinh, Haas, Otto, & Pankanti (2014), the authors describe a system for detecting tie plates and spikes. Their method, which is described in more detail in Trinh, Haas, Li, Otto, & Pankanti (2012), uses an AdaBoost-based object detector as seen in Viola & Jones (2001) and employs a model selection mechanism which assigns the object class that produces the highest number of detections within a window of 50 frames.

Recent advances in CMOS imaging technology have led to commercial-grade line-scan cameras that can capture images with high resolution and line rates of up to 140 KHz. High-intensity LED-based illuminators are available with life expectancies in the range of 50,000 hours providing nearly maintenance-free operation over several months. Therefore, technology that enables autonomous visual track inspection from an unattended vehicle (such as a passenger train) may become a reality in the not-too-distant future. Now that the systems integration challenges are solved, we expect that there will be a surge in applications in the near future.

1.2 Objectives

The goals of this project were two-fold:

1. Research novel computer vision techniques for detection of flaws in railway images, and
2. Develop these algorithms in software and demonstrate them with real data.

1.3 Overall Approach

This project aimed to facilitate the adoption of the newly-developed technology by the railroad industry. Technology transfer occurred during this project, including proof-of-concept code and usable test software. The system was evaluated by our industry partners, so more specific requirements could be gathered and future deployments could be facilitated. This process will facilitate future integration with systems used by the industry, thereby avoiding delays in technology deployment. To ensure a smooth technology transfer, we followed industry software development practices to guarantee code modularity, maintainability, verifiability and reproducibility.

1.4 Scope

High Speed Rail (HSR) track anomalies are harder to find than conventional track anomalies because more frequent inspections are required and shorter maintenance windows are available. Machine vision technology is being adopted by the industry to complement other forms of inspection.

This research effort is designed to provide the rail industry with the latest advances in vision-based anomaly detection and machine learning algorithms, and existing as well as new algorithms have been used during this project. It produced a prototype software module that takes images of rail components as its input and returns the position, size and type of each detected anomaly with a corresponding score. This research has been tested for three safety applications of special concern to high speed and intercity passenger rail inspectors:

1) Detection of cracks on concrete ties.
2) Detection of missing and broken rail fasteners.
3) Detection of crumbled and chipped concrete ties.

1.4.1 Dataset

The algorithms' performance has been demonstrated using data collected from concrete tie track and the tools have been designed to be relatively user-friendly.

This anomaly detection module has been demonstrated with CTIV, a visual track inspection system that is used by FRA, ENSCO, and the rail industry. It was used to collect the data used in this report and the FRA project "Concrete Tie Degradation Assessment." The images were collected at a resolution of 0.43 mm/pixel and a single color channel at 8 bits per pixel. ENSCO provided raw images and the output of their tie detection algorithm, while Amtrak provided metadata such as tie installation year, tie manufacturer, track speed, curvature, and annual tonnage. All the ties in this dataset are made of reinforced concrete, and they were manufactured by San-Vel or Rocla then installed between 1978 and 2010. The collected images were

automatically stitched together and saved into several files, each containing a 1-mile image. The dataset is summarized in Table 1.

Table 1. Dataset summary.

Survey	Date	# Miles	Data Size	# Ties	# Ties w/ conditions	# Ties w/ full annotations
BAA_Test1	**August 2012**	97	3.5 TB	236,578	25,314	5,005
BAA_Test2	**April 2013**	112	4.0 TB	281,040	2,150	1,101
BAA_Test3	**Sept. 2013**	120	3.7 TB	338,730	2,008	1,881

Since only a subset of the data was fully reviewed by ENSCO personnel during the tie degradation assessment, in this report we publish results based on that subset. We selected all the miles from the first 2 surveys that were manually aligned by ENSCO reviewers. This subset contains 85 miles of continuous trackbed images. Then we verified that all the tie boundaries in this subset were accurate after correcting invalid tie detections visually. Table 2 summarizes the data subset that was used for our experiments in Sections 5 and 6.

Table 2. Data subset used in our experiments.

MP range	Track #	Surveys
189	1	1, 2
191-194	1	1, 2
198-199	1	2
201	1	2
203	1	2
207-209	1	2
210-211	1	1, 2
212-213	1	2
143-144	2	1, 2
159	2	2
160-161	2	1, 2
162	2	2
163	2	1, 2
164-167	2	2
170-171	2	1, 2

MP range	Track #	Surveys
173	2	1, 2
176-177	2	1, 2
181-184	2	1, 2
186	2	1, 2
190-191	2	1, 2
192	2	1, 2
193-197	2	2
198	2	1, 2
199-203	2	2
205	2	2
206	2	1, 2
207-208	2	2
209	2	1, 2
210-211	2	2

1.5 Organization of the Report

The rest of this report is organized as follows:

- Section 2 describes the general design considerations and the system architecture
- Section 3 describes the front-end module (the Vision Client)
- Section 4 describes the crack detection algorithm
- Section 5 describes the fastener inspection algorithm
- Section 6 describes the material identification and chip/crumbling detection algorithm
- Section 7 discusses the conclusions of this work and potential future research directions
- Section 8 provides references to related material

2. System Architecture

This section describes the overall architecture of our system.

2.1 General Design Considerations

In this project, we designed a data processing and manipulation system that extracts information from large amounts of visual data. To ensure that the software meets current and future requirements, while minimizing design and implementation risks, the design has the following characteristics:

- **Simplicity:** If there are multiple approaches to implementing a feature, the least complex approach to meeting the requirements shall be selected. Simple interfaces based on standard protocols shall be exposed to developers and we shall prefer external libraries that provide a simple and consistent interface.
- **Data locality:** Data processing shall be scheduled to minimize the need of large data transfers. For instance, all the operations on a single piece of data (such as an image) should be performed on the same node, and intermediate results should be reused locally. We can still take advantage of parallel processing on the same image, but this parallelism should be limited to using multiple threads on the same node, and not by using separate nodes. When GPU hardware is used to perform computations, all processing steps shall be performed by the same GPU device to avoid time-consuming transfers of intermediate results. However, local memory transfers between host and GPU are still faster than remote transfers so any CPU processing that needs to be done on GPU-generated results should be done in the same node, even if there are other nodes with faster CPUs.
- **Throughput vs. Latency:** For the applications that are envisioned in this project, achieving maximum average throughput is more important than minimizing worst-case latency. Therefore, the scheduling will be designed for maximum resource utilization except for user interface threads, which will be granted higher scheduling priority than processing threads.
- **Accuracy vs. Speed:** For research purposes, the results must be exactly repeatable. Therefore, we will only select an optimization strategy that generates the same output for the same input. Thus, if there are algorithms that could be executed on either CPU or GPU and these algorithms involve floating point operations, their scheduling should be deterministic and for a given image, such operation must be performed by the same type of device, since different platforms use different numbers of significant digits and rounding methods. Also, since processors have internal floating-point registers that have higher precision than the operands, different processors may generate different results. The results are not guaranteed to be the same even on the same processor if different compiler settings are used, because certain math functions that can be compiled to a 64-bit binary can produce different results in the corresponding 32-bit binary. This is also the case across CUDA computer capabilities. For example, code running on a GPU with compute capability 2.0 takes advantage of optimizations that are not available on a device with compute capability 1.1, so results are not guaranteed to be the same. Therefore, the only way to ensure that the results are always the same is to run the same binary on the same processor type. Although this consideration is important during research, these differences are not statistically significant, so in practical deployments it will be possible to enable such optimizations.

- **Resource utilization:** To avoid processing bottlenecks, the scheduler shall use load balancing strategies to ensure that resource utilization across the cluster is almost constant.
- **Handover:** The software should support dynamic scheduling of processes across the cluster. Since the computing cluster is shared with other projects, jobs can only be scheduled for a limited length of time (wall time). Therefore, the process scheduler will be required to "reroute" tasks whenever a process terminates or a new process is launched to keep the processing flow in place.
- **Reproducibility:** The software should produce exactly repeatable results. This means that any algorithm that relies on random numbers should use a pseudorandom number generator that has been initialized with a deterministic seed which only depends on the input data, and the state of the generator should not be shared across threads. Also, all parallel paths should have synchronized merging, and all data transfers should be atomic.
- **Testability:** To facilitate debugging, the software shall provide several intermediate test points, so that for any error condition that may arise, we can easily build a unit test to help debug and fix it.
- **Error handling:** For research purposes, an error that leads to an undetermined state, such as data corruption, shall be considered a bug and shall immediately abort the data processing and notify the user. Timeouts shall also be considered bugs and shall also be treated as unrecoverable errors.
- **Logging:** The framework shall support centralized logging of events.

The following considerations will arise whenever we want to deploy this software into a real visual track inspection vehicle:

- **Causality:** If all the data is available, it is possible to infer a better decision if the system analyzes all the data at once (batch processing). However, in practice, the user does not want to wait until all the data is collected but he or she usually prefers to get results as soon as possible. Therefore, the system will have to provide results based on data collected up to a certain time. This sequential processing will be implemented with a fix delay, so it is possible to use some data "ahead" of the current location.
- **Direction independence:** Track inspection systems are usually expected to provide the same results independently from the direction the vehicle travelled then the data was acquired. This symmetry requirement, combined with the causality requirement implies that the inference and the decision must be performed using only data within a fixed window length. Although this symmetric mode of operation may make sense in an unattended setting, if we allow the system to learn from user feedback, the results will no longer be direction independent. Therefore, we will not take this constraint into consideration.

2.2 Software design

2.2.1 Programming languages

We have used C++ for all the code in this project. We have taken advantage of Qt's C++ extensions, and used Qt's signal/slot mechanism for event driven code. This code needs to be preprocessed with **moc**, the meta-object compiler provided with the Qt framework, but **moc** is

available for all major OSs so this is not a limitation. We have used MATLAB to quickly prototype some algorithms, but we have ported them to C++ due to the throughput limitations of MATLAB code. GPU code has been written in CUDA C++, which is an extension of the C++ programming language. CUDA C++ needs to be compiled with NVIDIA's **nvcc** compiler. The **nvcc** compiler is also available for all major OSs.

2.2.2 Software libraries

We have used third party libraries to speed up software development. To guarantee that the software is portable, we have given preference to libraries that are open-source and are available under a public license. For functionality that is not available from open-source libraries, we have used libraries that are well supported by the vendor, have a large user base, and are available under both Linux and Windows. We avoided using any software component that requires any payment of per user or per CPU royalties because that would have limited our ability to transfer the technology. Fortunately, there are plenty of high-quality libraries that meet these requirements. Table 3 enumerates all the third party libraries and frameworks that we have used in this project.

Table 3. Libraries used in this project.

Library	Description and Purpose	Version	License
Qt	Portable application framework, user interface, messaging and threading	5.3.1	LGPL
OpenCV	Image processing and computer vision	2.4.9	BSD
BVLC Caffe	Deep learning with convolutional neural networks	Rc	BSD
CUDA SDK	Parallel computing on GPU	6.5.14	Freeware
Intel IPP	Optimized image processing primitives	2013 sp1.1.106	Commercial

2.3 Modular architecture

The software for this project has four main modules. These modules have been designed to interface with each other, but the interface is open. In the future, this will facilitate the addition of new modules or the reconfiguration of existing ones, enabling new applications. The modules have generic components, so that the basic functionality can be reused to create specialized modules for specific applications such as tie crack detection, missing fastener detection or crumbling tie detection. In this section we only describe the generic functionality of each module. Figure 1 shows a potential system configuration using the components described in this report.

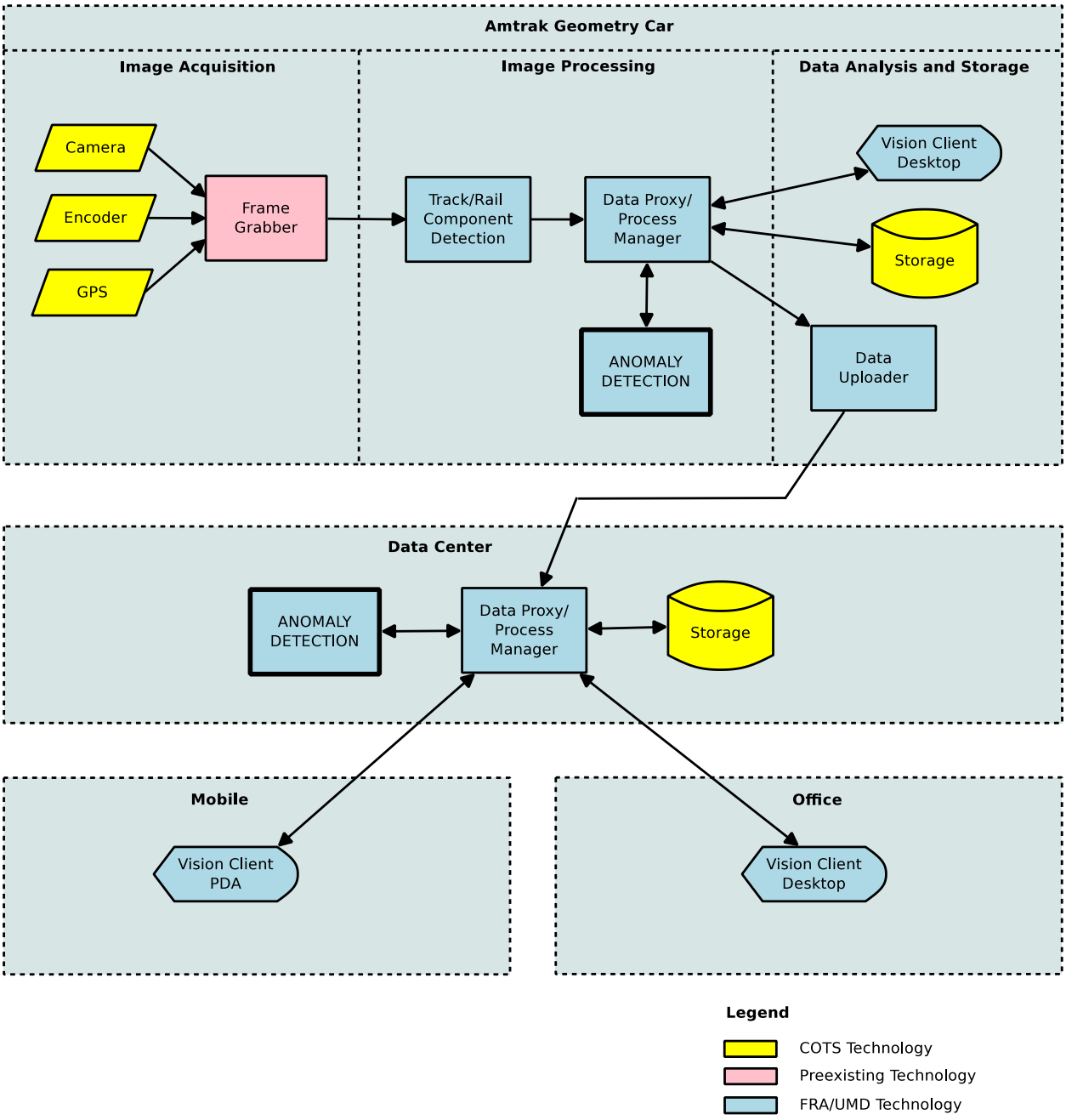


Figure 1. System configuration as envisioned to be deployed on a track inspection vehicle

2.3.1 Vision Client

This module is the front-end of the framework and it contains the user interface. The Vision Client can be used to view data, edit annotations, inspect intermediate results, and launch processing tasks. Section 3 provides more details about the client.

Table 4. Vision Client module overview.

Inputs	Interactive user input, data from backend
Outputs	Control messages to backend, user-generated data
Responsibilities	Data annotation, data review, launching of processing, monitoring
Interactions	Data proxy, process scheduler
Preconditions	It depends on the data operation performed by the user
Postconditions	Annotations will be updated (if the user has modified them)

2.3.2 Data Proxy

The data proxy allows all other components to communicate with each other. It listens to two different ports: one implements the HTTP protocol to serve data objects and data files and the other provides event notifications. The HTTP server supports both GET and POST commands. The main function of the data proxy is to serve as a repository of data objects. This server also accepts requests for raw data in case the applications cannot directly access the data. The data is fetched from the shared repository and is transferred via HTTP. We have configured a local Apache web server as a reverse proxy to reroute HTTP traffic over HTTPS, so the data is always transferred securely through the Internet.

Table 5. Data proxy overview.

Inputs	Data requests from other modules
Outputs	Data replies, event notifications
Responsibilities	Serve as a repository of data objects. Serve as an interface to the data, so other modules can safely access it. Send notifications to modules
Interactions	All other modules
Preconditions	None
Postconditions	Always reachable

2.3.3 Process Scheduler

The process scheduler abstracts the structure of the computing cluster for the rest of the system. The scheduler is responsible for launching, monitoring and terminating processes that run on several computers across the network. In our current implementation, the process scheduler is integrated within the data proxy.

Table 6. Process scheduler overview.

Inputs	Commands from front end, status from processing modules, event notifications from data proxy
Outputs	Commands to processing modules, commands to Torque
Responsibilities	Launch processing modules, schedule processing operations on the data, send commands to processing modules
Interactions	All other modules
Preconditions	Data proxy is running
Postconditions	Processing modules are running/stopped

2.3.4 Anomaly detection

This module runs image processing and image understanding algorithms. The anomaly detection module offers one or more of the following capabilities depending on how the command line arguments and compilation options are set:

- Preprocessing
- Crack detection
- Fasteners training
- Fasteners testing
- Texture testing

Table 7. Anomaly Detection module overview

Inputs	Data object containing raw images or preprocessed data
Outputs	Data object containing detection results or processed data
Responsibilities	Run algorithms on data
Interactions	Data proxy, process scheduler
Preconditions	Raw data or object containing preprocessed data has been posted
Postconditions	Preprocessed data object is created

3. Vision Client

In this section, we describe the front-end to the anomaly detection framework.

3.1 Overview

The UMD Railway Vision Client (the client) provides the user interface to the railway data repository. The purpose of the client is to:

1) Visualize the results of anomaly detection algorithms and validate or reject automated detections,

2) Generate ground truth data to be used to train such algorithms,

3) Provide a user interface for active learning (user-assisted) algorithms

The client can currently:

- Access the data repository
- Provide user authentication and encryption for secure data access
- Index images
- Display images
- Scroll through continuous 1 mile images
- Insert/remove/edit defect bounding boxes
- Provide pixel-level annotation of conditions
- Generate a side-by-side display of data from multiple surveys
- Transfer defect annotations between surveys
- Estimate of crack width/height/area and analysis of crack growth rates
- Query tie conditions with ability to save and retrieve user queries
- Query automatically detected fastener conditions
- Query automatically detected crumbling and chipped ties
- Filter queries to a subset of surveys and/or type of track
- Navigate quickly through search results
- Export filtered or unfiltered component lists including multiple surveys
- Export defect lists including multiple surveys
- Export filtered or unfiltered fastener lists with location, fastener type and fastener condition
- Insert/remove/edit tie bounding boxes
- Automatically align tie images between different surveys
- Provide multiplatform support (Windows/Linux/Mac/iOS)

In Figure 2, the client displays a broken tie. The client's left panel provides a hierarchical view of all the images in the repository, the right panel provides an aerial view of the track that is being inspected, and the bottom panel provides a view of the whole tie with metadata, while the central panel provides a close up of the currently selected defect and allows the user to zoom and scroll. This view can scroll through one mile of data.

The software contains menu options that allow the user to access specialized functions. For example, Figure 4 shows the interface for annotating fastener types to train the fastener detection algorithm, Figure 3 shows the interface for reviewing the results of fastener detection, and Figure 5 shows the effects of material classification and semantic segmentation on a crumbling tie. The Vision Client also supports twin mode (Figure 5), where images of the same tie from different surveys can be compared side-by-side. These are just a few examples of the capabilities offered by the Vision Client.

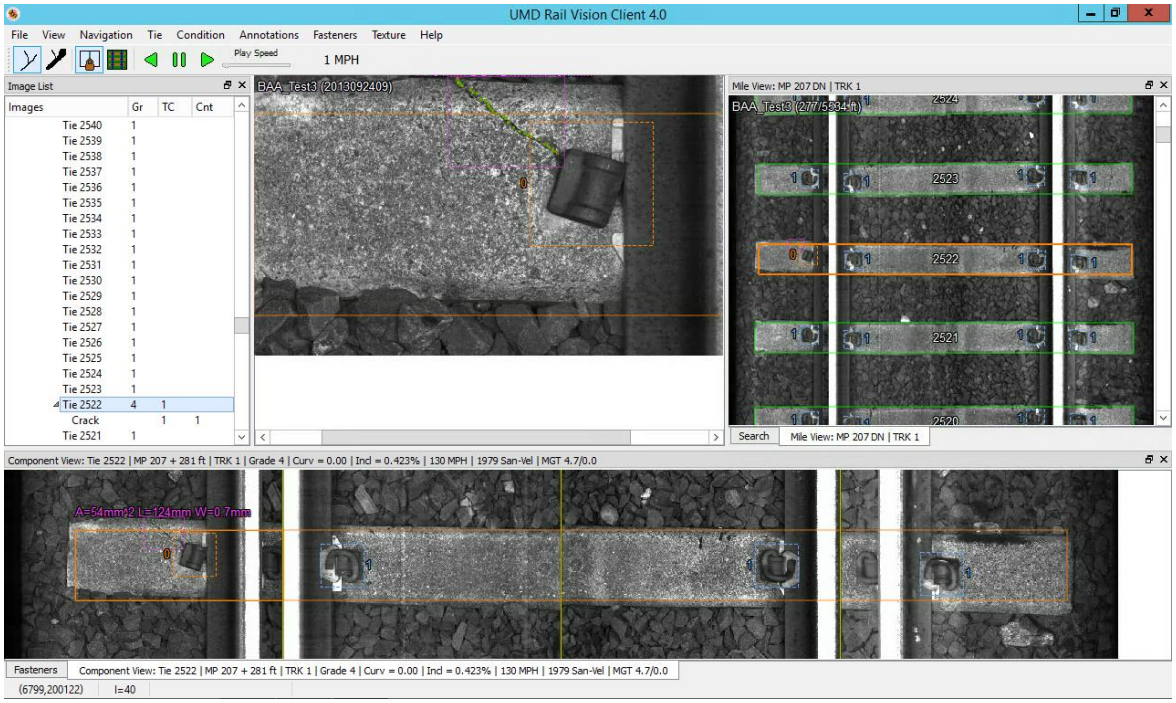


Figure 2. Default screen layout of the Railway Vision Client.

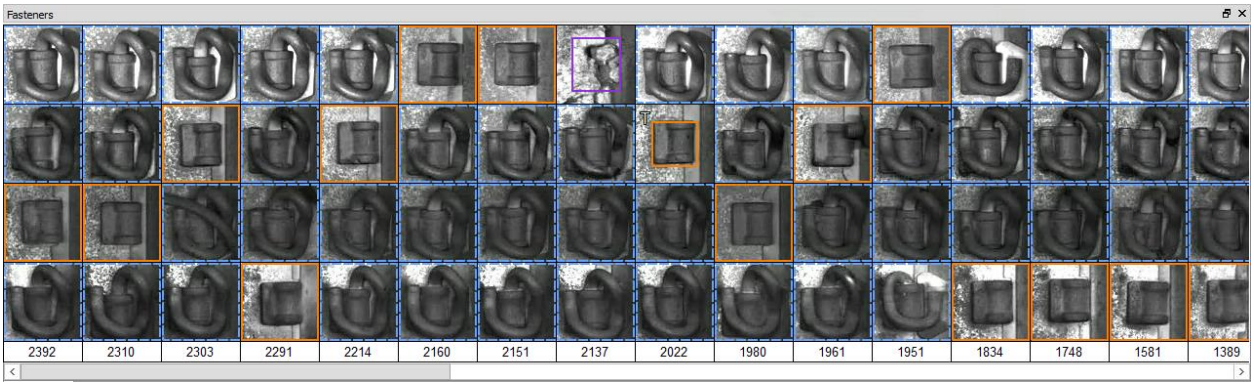


Figure 3. Fastener assessment results viewer.

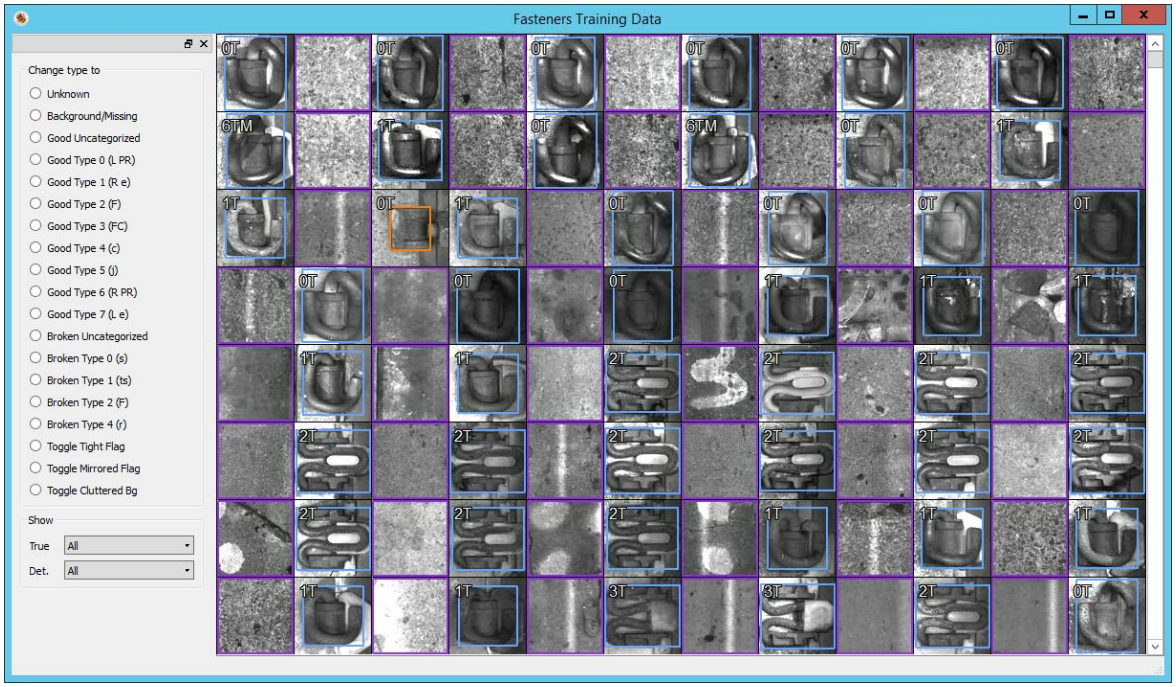


Figure 4. Fasteners training set review tool.

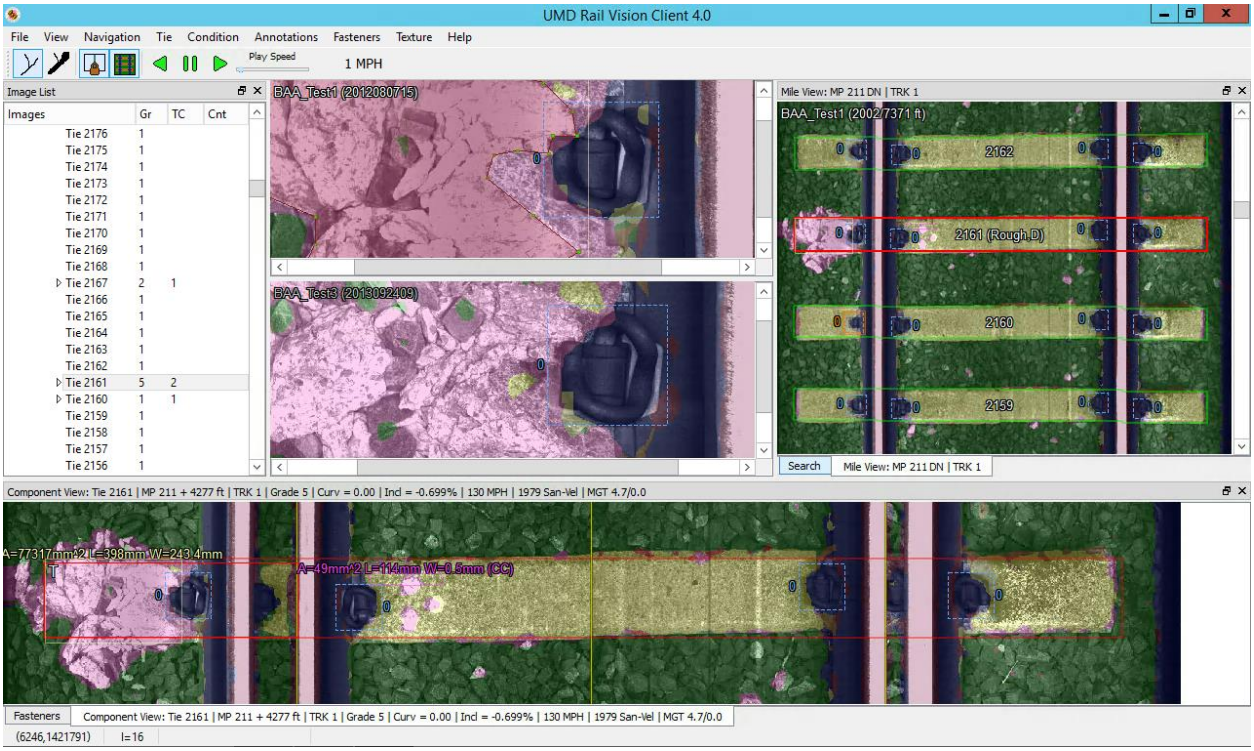


Figure 5. Railway Vision Client in twin view mode with fastener detection and semantic segmentation results overlaid.

4. Crack Detection Module

In this section, we describe the algorithm for detecting cracks and extracting crack measurements such as length of crack centerline and average crack width. Our crack detector is based on the Discrete Shearlet Transform (DST). Shearlets have emerged in recent years as one of the most successful methods for the multiscale analysis of multidimensional signals. Unlike wavelets, shearlets form a pyramid of well-localized functions that are defined not only over a range of scales and locations, but also over a range of orientations and with highly anisotropic supports. As a result, shearlets handle the geometry of multidimensional data much more effectively than traditional wavelets, and this has been exploited in a wide range of applications from image and signal processing.

4.1 Background

Detecting cracks on concrete structures is a difficult problem, due to the changes in width and direction of the cracks as well as the variability in the surface texture. This problem has recently received considerable attention. Redundant representations, such as undecimated wavelets, have been extensively used for crack detection (Subirats, Dumoulin, Legeay, & Barba, 2006) (Chambon & Moliard, 2011). However, wavelets have poor directional sensitivity and detecting weak diagonal cracks can be difficult. To overcome this limitation, Ma, Zhao, & Hou (2008) proposed the use of the nonsubsampled contourlet transform (Cunha, Zhou, & Do, 2006) for crack detection. However, all these methods rely on the assumption that the background surface can be modeled as additive white Gaussian noise and this assumption leads to matched filter solutions. Real images textures are highly correlated and applying linear filters causes poor performance.

To address this limitation, we adopted a completely new approach to crack detection based on separating the image into morphological distinct components using sparse representations, adaptive thresholding and variational regularization. This technique was pioneered by Stark et al. (Starck, Elad, & Donoho, 2005) and later extended and generalized by many authors such as Bobin, Starck, Fadili, Moudden, & Donoho (2007), Easley, Labate, & Negi (2013), and Kutyniok & Lim (2011). In particular, we will use the Iterative Shrinkage Algorithm with a combined dictionary of shearlets and wavelets to separate cracks from background texture.

4.2 Algorithm Description

We model an image *x* containing cracks on textural background as a superposition of a crack component x_c with a textural component x_t:

$$x = x_c + x_t$$

Let Φ_1 and Φ_2 be the dictionaries corresponding to wavelets and shearlets, respectively. We assume that x_c is sparse in a shearlet dictionary Φ_1 and similarly x_t is sparse in a wavelet dictionary Φ_2. That is, we assume that there are sparse coefficients a_c and a_t so that $x_c = \Phi_1 x_c$ and $x_t = \Phi_2 a_t$. Then, one can separate these components from an x via the coefficients a_c and a_t by solving the following optimization problem:

$$(\hat{a}_c, \hat{a}_t) = argmin_{a_c, a_t} \lambda \|a_c\|_1 + \lambda \|a_t\|_1 + \frac{1}{2} \|x - \Phi_1 a_c - \Phi_2 a_t\|_2^2,$$

where for an *n*-dimensional vector *b*, the ℓ_1 norm is defined as $\|b\|_1 = \sum_i |b_i|$. This image separation problem can be solved efficiently using an *iterative shrinkage algorithm* proposed in (Kutyniok & Lim, 2011).

4.3 Experimental Results

In our experiments, we used symlet wavelets with four decomposition levels to generate Φ_2 and a 4-level shearlet decomposition with Meyer filters of sizes 80 × 80 on all four scales, eight directional filters on the first three scales, and 16 directional filters on the forth scale, to generate Φ_1. To assess the performance of the separation algorithm, we calculated the ROC curves for each image using the following detection methods:

a) *Shearlet-C*: This method takes advantage of the Parseval property of the shearlet transform and performs crack detection directly in the transform domain. We first decompose the image into cracks and texture components using Iterative Shrinkage with a shearlet dictionary and a wavelet one. Instead of using the reconstructed image, we analyze the values of the shearlet transform coefficients. For each scale in the shearlet transform domain, we analyze the directional components corresponding to each displacement and collect the maximum magnitude across all directions. If the sign of the shearlet coefficient corresponding to the maximum magnitude is positive, we classify the corresponding pixel as background, otherwise we assign the norm of the vector containing the maximum responses at each scale to each pixel and we apply a threshold.

b) *Shearlet-I*: We first decompose the image into cracks and texture components as described for the previous method. Then, we apply an intensity threshold on the reconstructed cracks image.

We compare our results to the following two basic methods not based on shearlets:

c) *Intensity*: This is the most basic approach, which only uses image intensity. After compensating for slow variations of intensity in the image, we apply a global threshold.

d) *Canny*: We use the Canny edge detector (Canny, 1986) as implemented in MATLAB using the default $\sigma = \sqrt{2}$ and the default high to low threshold ratio of 40%.

After using a low-level detector, it may be necessary to remove small isolated regions corresponding to false detections due to random noise. This postprocessing step may reduce the false detection rate on intensity-based methods. However, to provide an objective comparison, we have generated the experimental results without running any postprocessing. We leave the perfomance analysis of a complete crack detector for future work.

To evaluate the performance of each crack detector, we manually annotated the crack pixels in each image. To mitigate the effect of ambiguous segmentation boundaries, we annotated the boundaries around the cracks as tightly as possible (making sure that only pixels completely contained inside the crack boundaries are annotated as such) and defined an envelope region around each crack whose labels are treated as "do not care". Formally, let Ω denote the set of pixels in the image, and *F* (foreground) denote the set of pixels labeled as cracks. We define the set *B* (backgrond) as

$$B = \{x \in \Omega : \min_{f \in F} \|x - f\| > \delta\},$$

where $\|x - f\|$ denotes the Euclidean distance between sites x and f. In our experiments we used $\delta = 3$. To account for possible small inaccuracies in the ground truth, we performed a bipartite graph matching between the detected crack pixels and the crack pixels in the ground truth. For our experiments, we allow matching within a maximum distance of two pixels. This choice of matching metric does not penalize crack overestimation errors as long as these errors are contained in such envelope. This allows us to decouple errors in estimating the position of the crack centerline from errors in estimating the crack width, which is more sensitive to lighting variations. Let D be the set of pixels detected as cracks by a given detector and

$$tp = |D \cap F| \qquad fn = |\overline{D} \cap F| \qquad p = tp + fn = |F|$$
$$tn = |\overline{D} \cap B| \qquad fp = |D \cap B| \qquad n = tn + fp = |B|$$

The probability of detection (PD) and false alarm (PF) are defined as

$$PD = \frac{tp}{p} \qquad PF = \frac{fp}{n}$$

A sequence of admissible detectors $D|_{PF \leq \epsilon}$ for a given false alarm rate ϵ, would produce monotonically increasing detection rates, $PD|_{PF \leq \epsilon}$. The Receiver Operating Characteristic function (ROC curve) is defined as *PD* as a function of *PF*.

$$ROC(x) = \max_{\epsilon \leq x} PD|_{PF \leq \epsilon}$$

One commonly used metric is the Area Under the ROC Curve (AUC), defined by

$$AUC = \int_0^1 ROC(x)dx,$$

which corresponds to the probability that a sample randomly drawn from F will receive a score higher than a sample randomly drawn from B. AUC provides a measure of the average performance of the detection across all possible sensitivity settings. Although it is an important measure, practically we are interested in knowing how well the detector will work when we choose a particular sensitivity setting. For this reason, we have selected Constant False Alarm Rate (CFAR) detectors with PF = 10^{-3} and PF = 10^{-4} and we report the corresponding PD. For completeness, we also report the F1 score (also known as the Dice similarity index), which is defined as

$$F_1 = \frac{2\ tp}{2\ tp + fn + fp}$$

In this report, we report the peak F_1 score for all methods. The Canny edge detection method (Canny, 1986) estimates the location of the crack boundary, while the other three methods estimate the location of the crack itself. To have a meaningful comparison, we have generated a separate ground truth masks for the crack outline, so we can use the same matching metric on the Canny method. For each method, we have used the same algorithm parameters on all the images.

Figure 6 shows three example images of typical cracks on each type of background texture (coarse, medium and smooth). The images have been intensity-normalized, and cropped into patches of 512 × 512 pixels. This image size is dyadic, so that we can achieve fastest computation of the FFT-based Shearlet transform and is enough to cover the whole width of an

8" tie. Figure 7 shows qualitative crack detection results and Figure 8 shows the ROC curves for all tested methods.

Table 8 summarizes our results. We observe that our shearlet-based detectors perform consistently well on all evaluation metrics. Note that, on Image 3, the Shearlet-I method, which is based on intensity in the reconstructed image, produces better results than all other methods. Due to its simplicity, intensity-based methods are still being used. For example, the system recently proposed in Oliveira & Correia, 2013) uses pixel intensities to detect cracks on road pavement. Based on the results from Table 8, we can conclude that, with the proper image preprocessing, intensity can still be a powerful feature for crack detection. However, the detection performance provided by shearlet-based features is more consistent across images. In future work, we will further explore the potential of combining both intensity and shearlet-based features. With any of the methods described in this section, it may be possible to further remove small artifacts in the detected crack boundary by adding a postprocessing step as was done in (Chambon & Moliard, 2011).

Table 8. Comparison of detection performance for different crack detection algorithms.

Image	Method	AUC	F1 score	$PD_{PF=10^{-3}}$	$PD_{PF=10^{-4}}$
1	Shearlet-C	**0.99915**	**0.79916**	**0.8398**	**0.6746**
	Shearlet-I	0.99908	0.65810	0.7140	0.4247
	Intensity	0.99874	0.73188	0.7411	0.5722
	Canny	0.94457	0.27752	0.2114	0.1099
2	Shearlet-C	**0.99999**	**0.98841**	**0.9989**	**0.9895**
	Shearlet-I	0.99557	0.62705	0.4837	0.3964
	Intensity	0.99037	0.55404	0.4371	0.3342
	Canny	0.99043	0.81787	0.6425	0.4462
3	Shearlet-C	0.99934	0.76418	0.8368	0.5874
	Shearlet-I	**0.99977**	**0.82353**	**0.9101**	**0.7098**
	Intensity	0.99650	0.45992	0.0543	0.0000
	Canny	0.96248	0.19436	0.0000	0.0000

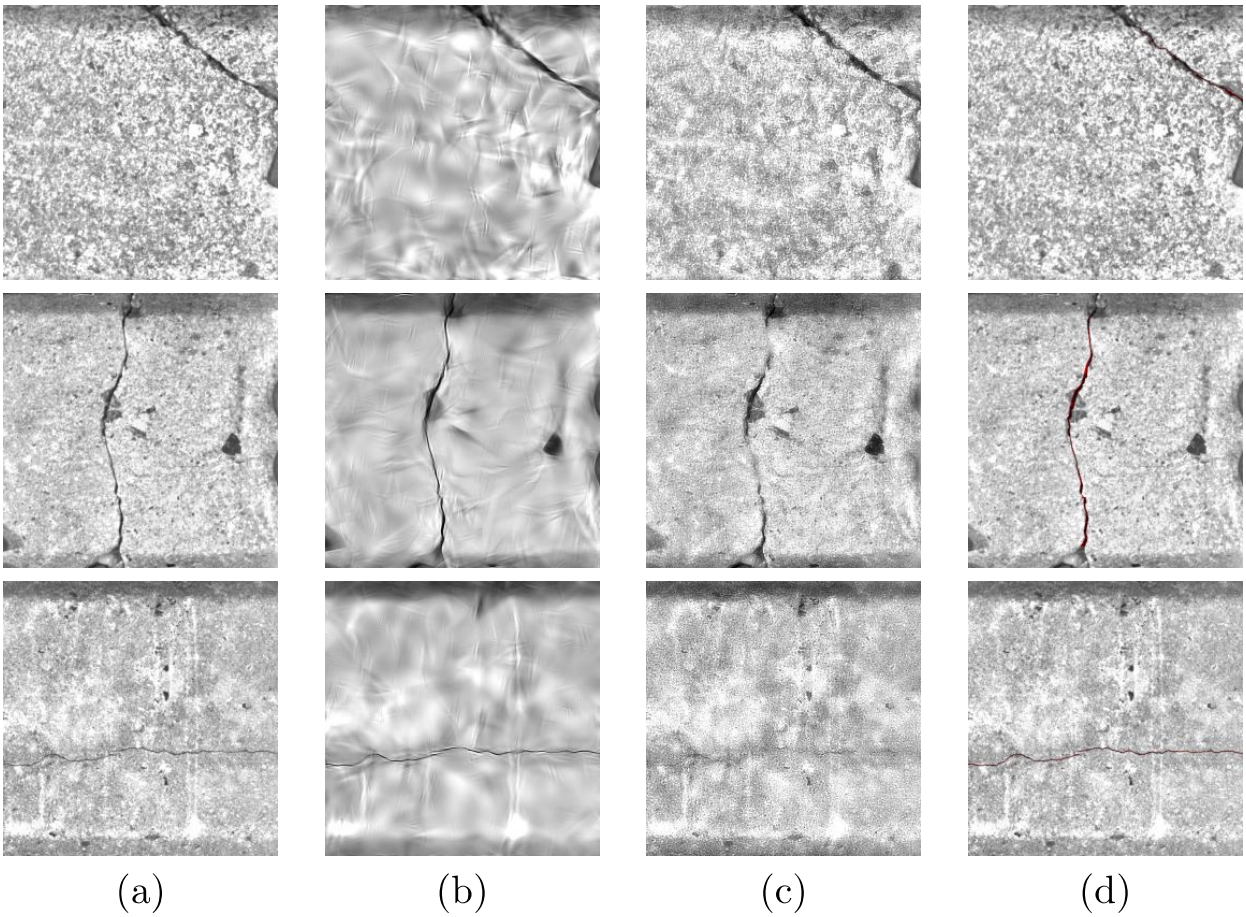


Figure 6. Image separation.

(a) Original images separated into **(b)** cracks and **(c)** textural background components, and **(d)** crack ground truth

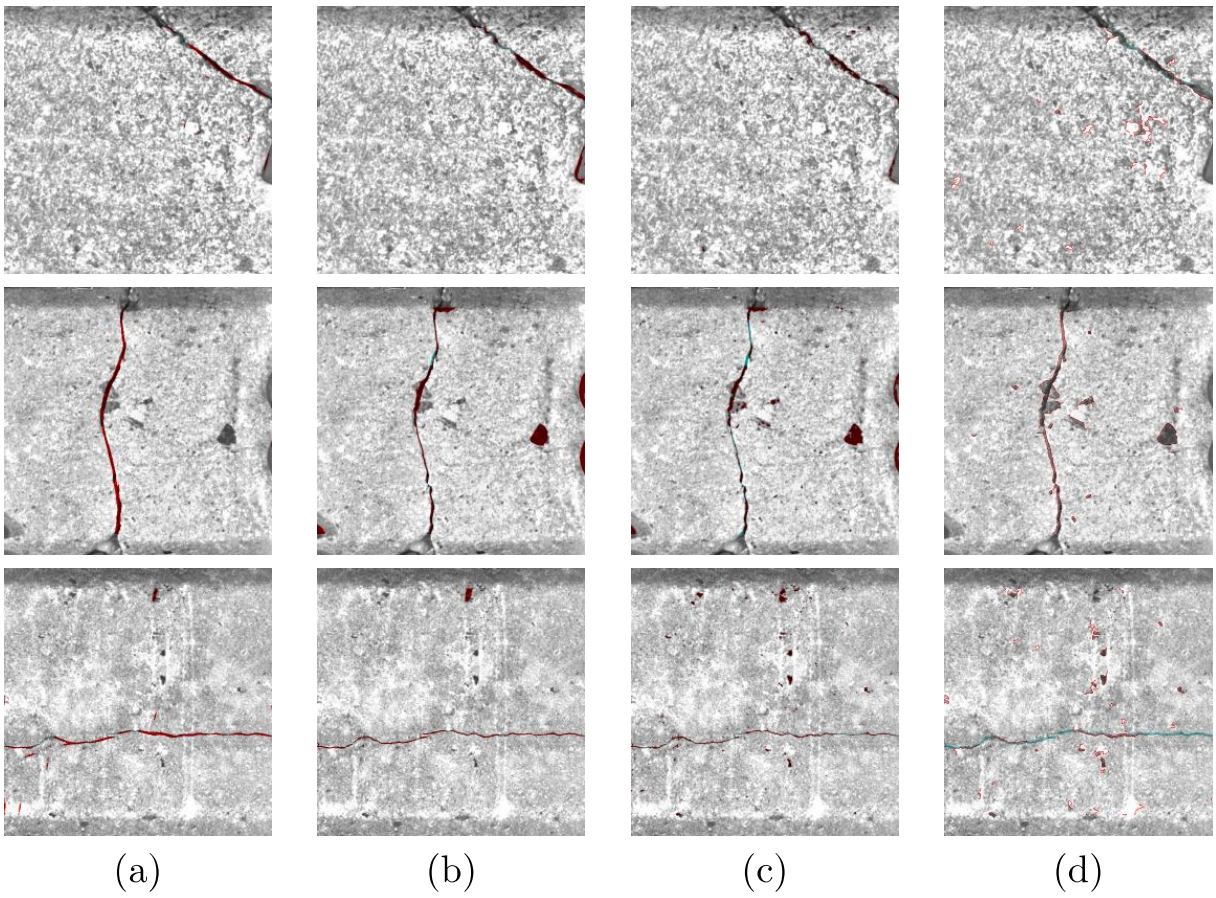


Figure 7. Crack detection results.

(a) Using shearlet coefficients (Shearlet-C) **(b)** using thresholding in the image reconstructed using shearlets (Shearlet-I) **(c)** using intensity thresholding in the original image, and **(d)** using Canny edge detection.

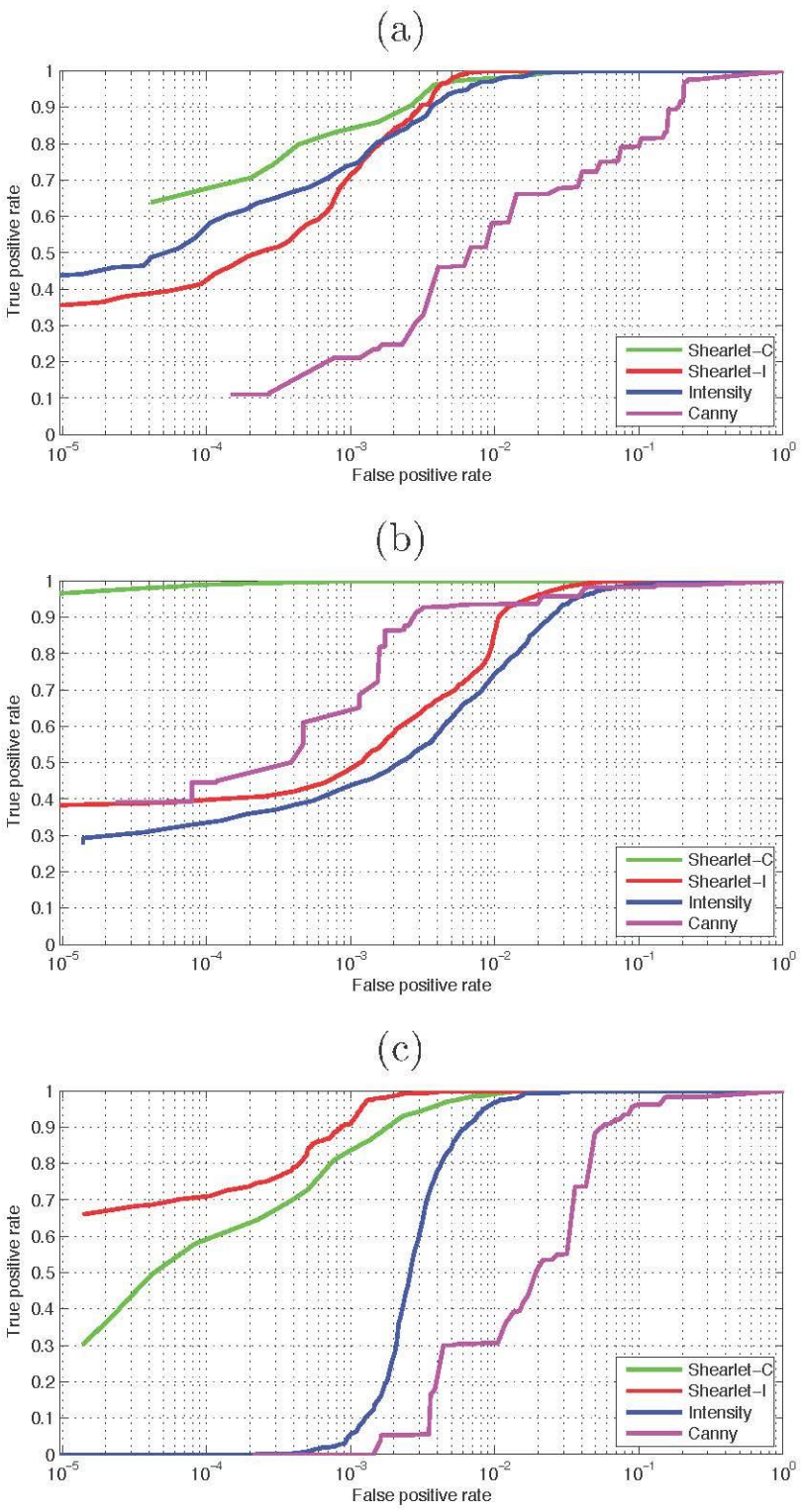


Figure 8. ROC curves for crack detection.

(a) Image 1, **(b)** image 2, **(c)** image 3.

5. Fasteners Inspection

Fasteners maintain the rails in a fixed position and they are critical railway components. If they fail, train derailments will occur due to gage widening or wheel climb, so their condition needs to be periodically monitored. Several computer vision methods have been proposed in the literature for track inspection applications, but these methods either require laser ranging or are not robust enough to deal with the clutter and background noise that is present in the railroad environment. In this section, we demonstrate that it is possible to inspect tracks for missing and broken fasteners with computer vision techniques that only use grayscale images and need no additional sensors. We have achieved this by 1) carefully aligning the training data, 2) reducing intra-class variation, and 3) bootstrapping difficult samples to improve the classification margin. Using the histogram of oriented gradients features and a combination of linear SVM classifiers, the algorithm described in this section can inspect ties for missing or defective rail fastener problems with a probability of detection of 98% and a false alarm rate of 1.23%.

5.1 Approach

In this section, we describe the details of our proposed approach to automatic fastener detection. Figure 9 shows the types of defects that our algorithm can detect. The detectors have been tested on concrete ties, but the framework can easily accommodate other types of fasteners and ties.

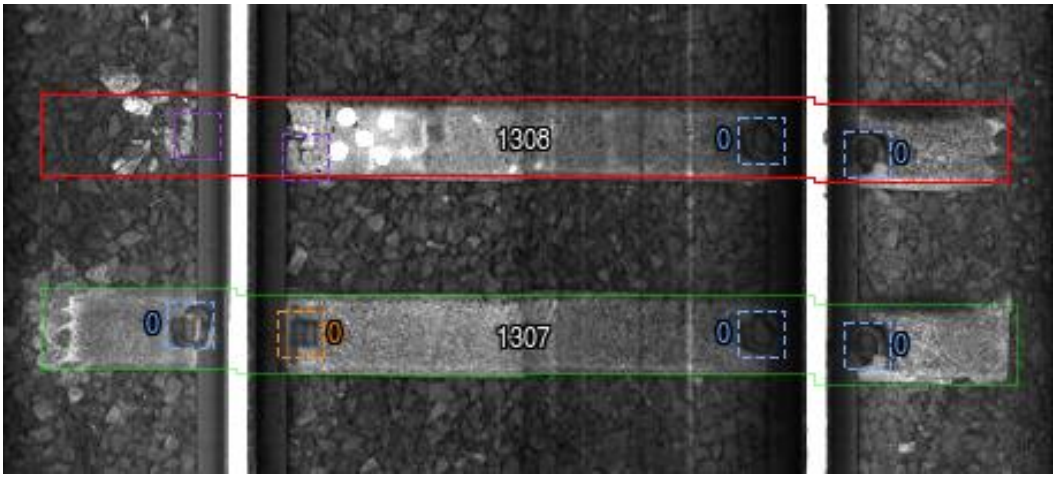


Figure 9. Example of defects that our algorithm can detect.

Blue boxes indicate good fasteners, orange boxes indicate broken fasteners, and purple boxes indicate missing fasteners. White numbers indicate tie index from last mile post. Other numbers indicate type of fastener (for example, 0 is for e-clip fastener).

5.1.1 Overview

Due to surface variations that result from grease, rust and other elements in the outdoor environment, segmenting railway components is very difficult. Therefore, we avoid that task by using a detector based on a sliding window that we run over the "inspectable" area of the tie. The

detector uses the well-known descriptor based on the Histograms of Oriented Gradients (HOG) (Dalal & Triggs, 2005), which was originally designed for pedestrian detection, but it has been proven effective for a variety of object detection tasks in unconstrained environments. Though fasteners are usually located very close to the rail, we need to search over a much broader area because on turnouts (switches and frogs) fasteners are positioned farther away from the rail, with more varied configurations.

5.1.2 Classification

Our goal is to simultaneously detect the most likely fastener location within each predefined Region of Interest (ROI), and then classify such detections into one of three basic conditions: background (or missing fastener), broken fastener, and good fastener. Then, for good and broken fastener conditions, we want to assign class labels for each fastener type (PR clip, e-clip, fastclip, c-clip, and j-clip).

Figure 10 shows the complete categorization that we use, from coarsest to finest. At the coarsest level, we want to classify fastener vs. unstructured background clutter. The background class also includes images of ties where fasteners are completely missing because: 1) it is very difficult to train a detector to find the small hole left on the tie after the whole fastener has been ripped off, 2) we do not have enough training examples of missing fasteners, and 3) most missing fasteners are on crumbled ties for which the hole is no longer visible.

Once we detect the most likely fastener location, we want to classify the detected fastener within the broken vs. good spectrum and then classify it into the most likely fastener type. Although this top-down reasoning works for a human inspector, it does not work accurately in a computer vision system because both the background class and the fastener class have too much intra-class variations. As a result, we employ a bottom-up approach.

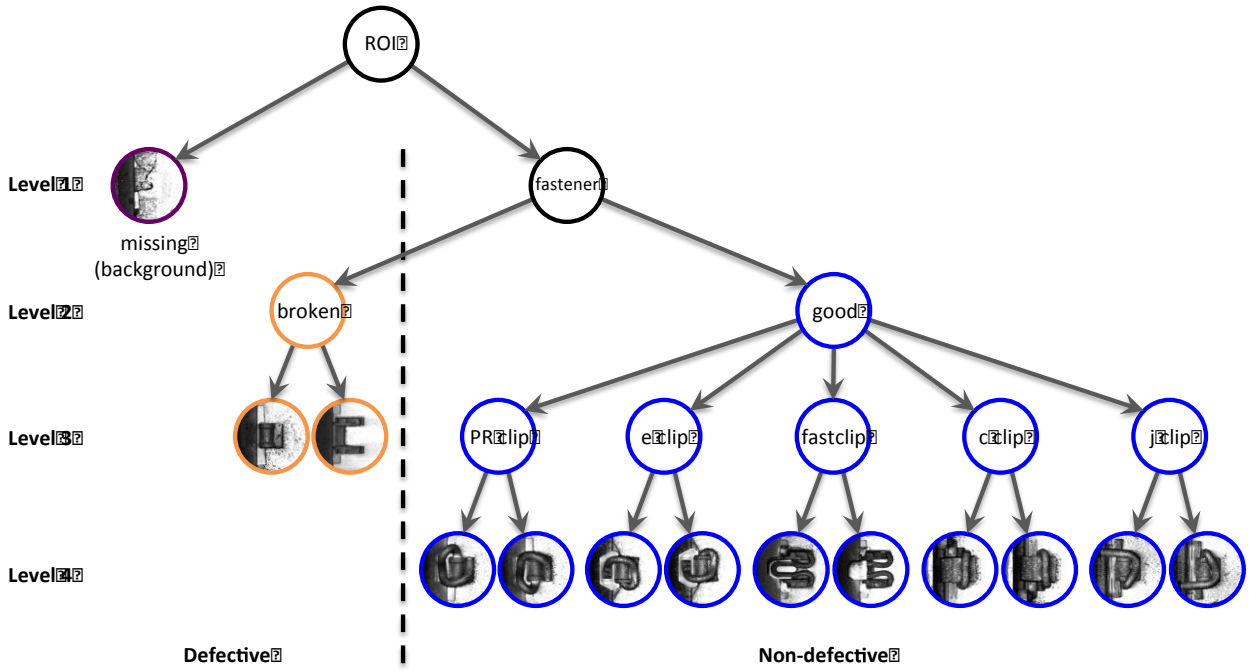
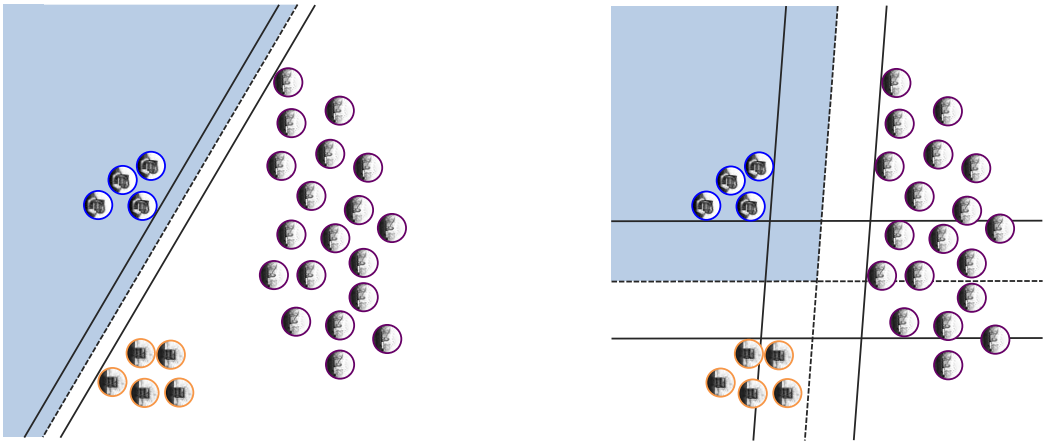


Figure 10. Object categories used for detection and classification (from coarsest to finest levels).

Since we use inner products, our detector may resemble the correlation-based approach used in (Babenko, 2009), but there are three key differences that set us apart: 1) our input is a HOG feature vector rather than raw pixel intensities, 2) we use a linear SVM to learn the coefficients of the detection filter, 3) we use a second classifier to reject misclassified fastener types.



(a) (b)

Figure 11. Justification for using two classifiers for each object category.

Shaded decision region corresponds fastener in good condition, while white region corresponds to defective fastener. Blue circles are good fasteners, orange circles are

broken fasteners, and purple circles are background/missing fasteners. (a) Classification region of good fastener vs. rest (b) Classification region of intersection of good fastener vs. background and good fastener vs. rest-minus-background. The margin is much wider than using a single classifier.

To achieve the best possible generalization at test time, we have based our detector on the maximum margin principle of the SVM. Although SVM is a binary classifier, it is straightforward to build a multi-class SVM, for example, by combining several one-vs-rest or one-vs-one SVM classifiers, either by a voting scheme or by using the DAG-SVM framework (Platt, Cristianini, & Shawe-taylor, 1999). Our approach uses the one-vs-rest strategy, but instead of treating the background class as just another object class, we treat it as a special case and use a pair of SVMs per object class.

For instance, if we had used a single learning machine, we would be forcing the classifier to perform two different unrelated tasks: a) reject that the image patch that does not contain random texture and b) reject that the object does not belong to the given category. Therefore, given a set of object classes $\mathcal{C}$, we train two detectors for each object category. The first one, with weights b_c, classifies each object class $c \in \mathcal{C}$ vs. the background/missing class $c \notin \mathcal{C}$, and the second one, with weights f_c classifies object class c vs. other object classes $\mathcal{C} \backslash c$. As illustrated in Figure 11, asking our linear classifier to perform both tasks at the same time would result in a narrower margin than training separate classifiers for each individual task. Moreover, to avoid rejecting cases where all f_c classifiers produce negative responses but one or more b_c classifiers produce strong positive responses that would otherwise indicate the presence of a fastener, we use the negative output of f_c as a soft penalty. Then the likelihood that sample x belongs to class c becomes

$$L_c(x) = b_c \cdot x + \min(0, f_c \cdot x),$$

where $x = HOG(I)$ is the feature vector extracted from a given image patch I. The likelihood that our search region contains at least one object of class c is the score of the union, so

$$L_c = \max_{x \in \mathcal{X}} L_c(x),$$

where $\mathcal{X}$ is the set of all feature vectors extracted within the search region, and our classification rule becomes

$$\hat{c} = \begin{cases} \arg\max_{c \in \mathcal{C}} L_c & \max_{c \in \mathcal{C}} L_c > 0 \\ m & otherwise. \end{cases}$$

5.1.3 Score Calculation

For the practical applicability of our detector, it needs to output a scalar value that can be compared to a user-selectable threshold τ. Since there are several ways for a fastener to be defective (either missing or broken), we need to generate a single score that informs the user how confident the system is that the image contains a fastener in good condition. This score is generated by combining the output of the binary classifiers introduced in the previous section.

We denote the subset of classes corresponding to good fasteners as $\mathcal{G}$ and that of broken fasteners as $\mathcal{B}$. These two subsets are mutually exclusive, so $\mathcal{C} = \mathcal{G} \cup \mathcal{B}$ and $\mathcal{G} \cap \mathcal{B} = \emptyset$. To build the

score function, we first compute the score for rejecting the missing fastener hypothesis (i.e, the likelihood that there is at least one sample $x \in \mathcal{X}$ such that $x \notin m$) as

$$S_m = \max_{c \in \mathcal{G}} L_c$$

where L_c is the likelihood of class c as previously defined. Similarly, we compute the score for rejecting the broken fastener hypothesis (i.e, the likelihood that for each sample $x \in \mathcal{X}$, $x \notin \mathcal{B}$) as

$$S_b = -\max_{c \in \mathcal{B}} \max_{x \in \mathcal{X}} f_c \cdot x,$$

The reason why the S_b does not depend on a *c*-vs-background classifier b_c is because mistakes between missing and broken fastener classes do not need to be penalized. Therefore, S_b need only produce low scores when x matches at least one of the models in $\mathcal{B}$. The negative sign in S_b results from the convention that a fastener in good condition should have a large positive score. The final score becomes the intersection of these two scores.

$$S = \min(S_m, S_b).$$

The final decision is done by reporting the fastener as good if $S > \tau$, and defective otherwise.

5.1.4 Training Procedure

The advantage of using a maximum-margin classifier is that once we have enough support vectors for a particular class, it is not necessary to add more inliers to improve classification performance. Therefore, we can potentially achieve relatively good performance with only having to annotate a very small fraction of the data. To generate our training set, we initially selected ~30 good quality (with no occlusion and clean edges) samples from each object category at random from the whole repository and annotated the bounding box location and object class for each of them. Our training software also automatically picks, using a randomly generated offset, a background patch adjacent to each of the selected samples.

Once we had enough samples from each class, we trained binary classifiers for each of the classes against the background and tested on the whole dataset. Then, we randomly selected misclassified samples and added those that had good or acceptable quality (no occlusion) to the training set. To maintain the balance of the training set, we also added, for each difficult sample, 2 or 3 neighboring samples. Since there are special types of fasteners that do not occur very frequently (such as the c-clips or j-clips used around joint bars), in order to keep the number of samples of each type in the training set as balanced as possible, we added as many of these infrequent types as we could find. Figure 12 shows a subset of our training set for fastener detection and classification.

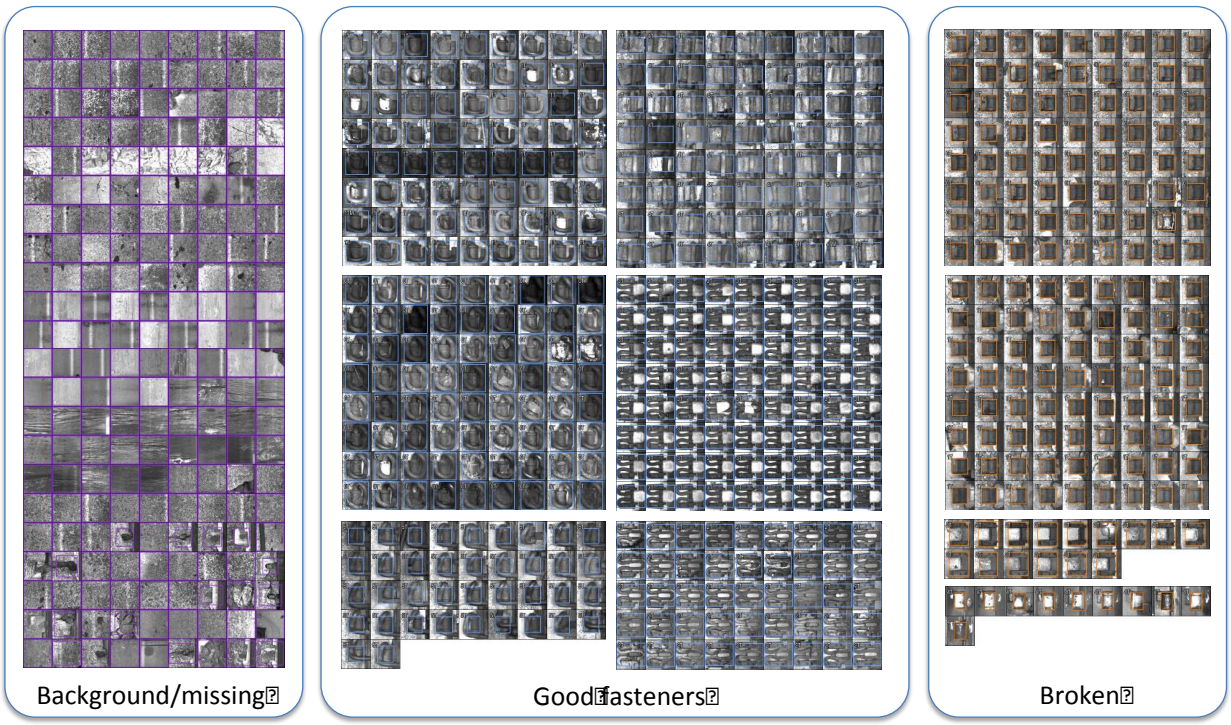


Figure 12. Examples of fastener images used to train our detector.

5.1.5 Alignment Procedure

For learning the most effective object detection models, the importance of properly aligning the training samples cannot be emphasized enough. Misalignment between different training samples would cause unnecessary intra-class variation that would degrade detection performance. Therefore, all the training bounding boxes were manually annotated, as tightly as possible to the object contour by the same person to avoid inducing any annotation bias. For training the fastener vs. background detectors, our software cropped the training samples using a detection window centered around these boxes and for training the fastener vs. rest detectors, our software centered the positive samples using the user annotation and the negative samples were re-centered to the position where the fastener vs. background detector generated the highest response. This was done to force the learning machine to learn to differentiate object categories by taking into account parts that are not common across object categories.

5.2 Experimental Results

To evaluate the accuracy of our fastener detector, we have tested it on the data subset introduced in Table 2. We downsampled the images by a factor of 2, for a pixel size of 0.86 mm. To assess the detection performance under different operating conditions, we flagged special track sections where the fastener visible area was less than 50% due to a variety of occluding conditions, such as protecting covers for track-mounted equipment or ballast accumulated on the top of the tie. We also flagged turnouts so we could report separate ROC curves for both including and excluding them.

5.2.1 Fastener Categorization

On our dataset, we have a total of eight object categories (two for broken clips, one for PR clips, one for e-clips,two for fast clips, one for c-clips, and one for j-clips) plus a special category for background (which includes missing fasteners). We also have four synthetically generated categories by mirroring non-symmetric object classes (PR, e, c, and j clips), so we use a total of 12 object categories at test time. The HOG features are extracted using a 160 × 160 pixel sliding window with a strap of 8 × 8. We use the HOG implementation in the object detection module of OpenCV using default parameters. For classification, we use the linear SVM implementation in the machine learning module of OpenCV (which is derived from *LIBSVM*) with a soft margin (C=0.01). Figure 13 shows an example of the HOG features extracted from a fastclip fastener.

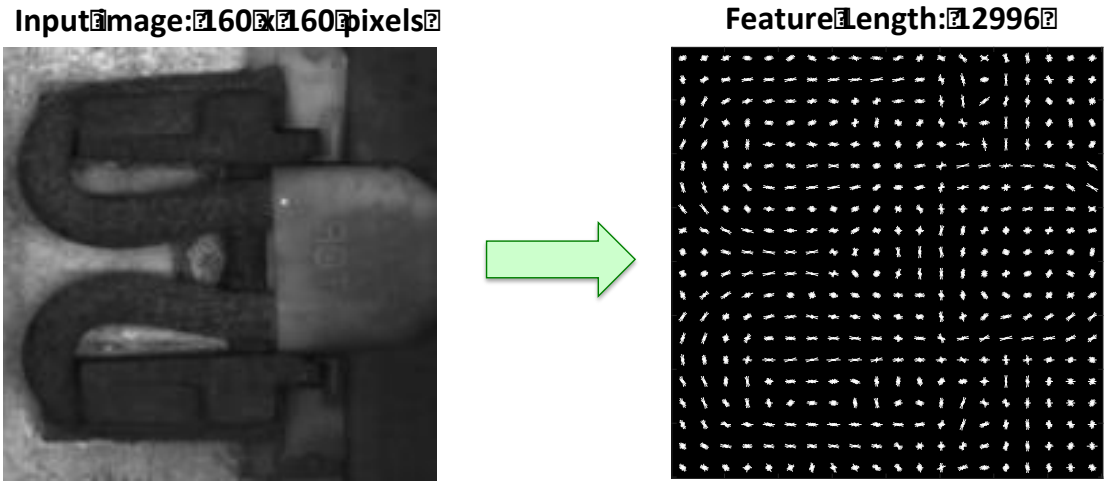


Figure 13. Feature extraction for fastener detection.

For training our detectors, we used a total of 3,805 image patches, including 1,069 good fasteners, 714 broken fasteners, 33 missing fasteners, and 1,989 patches of background texture. During training, we also included the mirrored versions of the missing/background patches and all symmetric object classes. To evaluate the feasibility of the algorithm, we performed 5-fold cross-validation on the training set, where we classified each patch into one of the nine basic object categories (we excluded the four artificially generated mirrored categories). Figure 14 (a) shows the resulting confusion matrix. We only had 14 misclassified samples (0.37% error rate). If we consider the binary decision problem of finding defective fasteners (either missing or broken), we have a detection rate of 99.74% with a false alarm rate of 0.65%. This is an encouraging result, since as explained in section 5.1.4, our training set has been bootstrapped to contain many difficult samples.

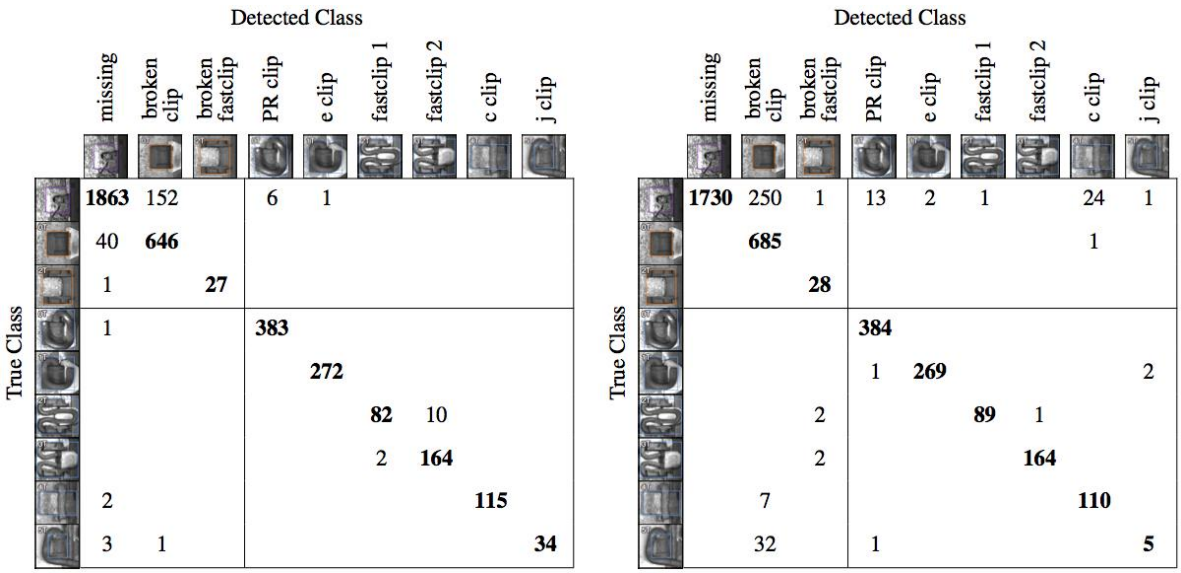


Figure 14. Confusion matrix on 5-fold cross-validation of the training set using (a) the proposed method (b) the method described in (Babenko, 2009) with HOG features.

In addition to the proposed method described in section 5.1, we have also implemented and evaluated the following alternative methods:

- **Intensity normalized OT-MACH**: As in (Babenko, 2009), for each image patch, we subtract the mean and normalize the image vector to unit norm. For each class *c*, we design an OT-MACH filter in the Fourier domain using $h_c = [\alpha I + (1-\alpha)D_c]^{-1}\,\overline{x_c}$ with $\alpha = 0.95$, where *I* is the identity matrix, $D_c = (1/N_c)\sum_{i=1}^{N_c} x_{ci}x_{ci}^*$, and N_c is the number of training samples of class c.
- **HOG features with OT-MACH**: The method in (Babenko, 2009), but replacing intensity with HOG features. Since HOG features are already intensity-invariant, the design of the filters reduces to $h_c = \overline{x_c}$.
- **HOG features with DAG-SVM**: We run one-vs-one SVM classifiers in sequence. We first run each class against the background on each candidate region. If at least one classifier indicates that the patch is not background, then we run the DAG-SVM algorithm (Platt, Cristianini, & Shawe-taylor, 1999) over the remaining classes.
- **HOG features with majority voting SVM**: We run all possible one-vs-one SVM classifiers and select the class with the maximum number of votes.

For the first and second methods, we calculate the score using the formulation introduced in sections 5.1.2 and 5.1.3, but with $b_c = h_c$ and $f_c = h_c - \sum_{i \neq c} h_i/(C-1)$. For the third and last methods, we first estimate the most likely class in $\mathcal{G}$ and $\mathcal{B}$. Then, we set S_b as the output of the classifier between these two classes, and S_m as the output of the classifier between the background and the most likely class.

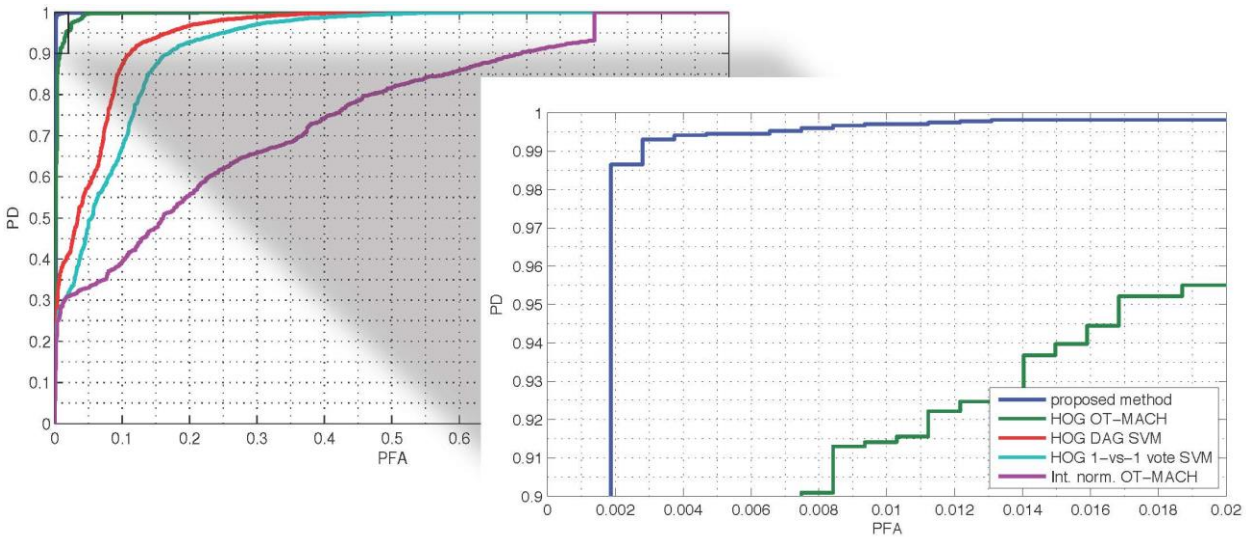


Figure 15. ROC curves for the task of detecting defective (missing or broken) fasteners using 5-fold cross-validation on the training set.

In Figure 15, we can observe that the proposed method is the most accurate, followed by the HOG with OT-MACH method. The other methods are clearly inferior. Figure 14 shows the confusion matrix of our method and the second best method. This method had an error rate of 2.23% (6 times greater than our proposed method). The detection rate was 98.86% with a false alarm rate of 4.02%. We can see that j-clips and c-clips are the most difficult types of fasteners because they contain more intra-class variation than other types; these fasteners are placed next to joint bars, so some of them are slightly rotated to accommodate the presence of joint bar bolts.

5.2.2 Defect Detection

To evaluate the performance of our defect detector, we divided each tie into four regions of interest (left field, left gage, right gage, right field) and calculated the score defined in section 5.1.3 for each of them. Figure 15 shows the ROC curve for cross-validation on the training set, and Figure 16 for the testing set of 813,148 ROIs (203,287 ties). The testing set contains 1,051 ties images with at least one defective fastener per tie. The total number of defective fasteners in the testing set was 1,086 (0.13% of all the fasteners), including 22 completely missing fasteners and 1,064 broken fasteners. The number of ties that we flagged as "uninspectable" is 2,524 (1,093 on switches, 350 on lubricators, 795 covered in ballast, and 286 with other issues).

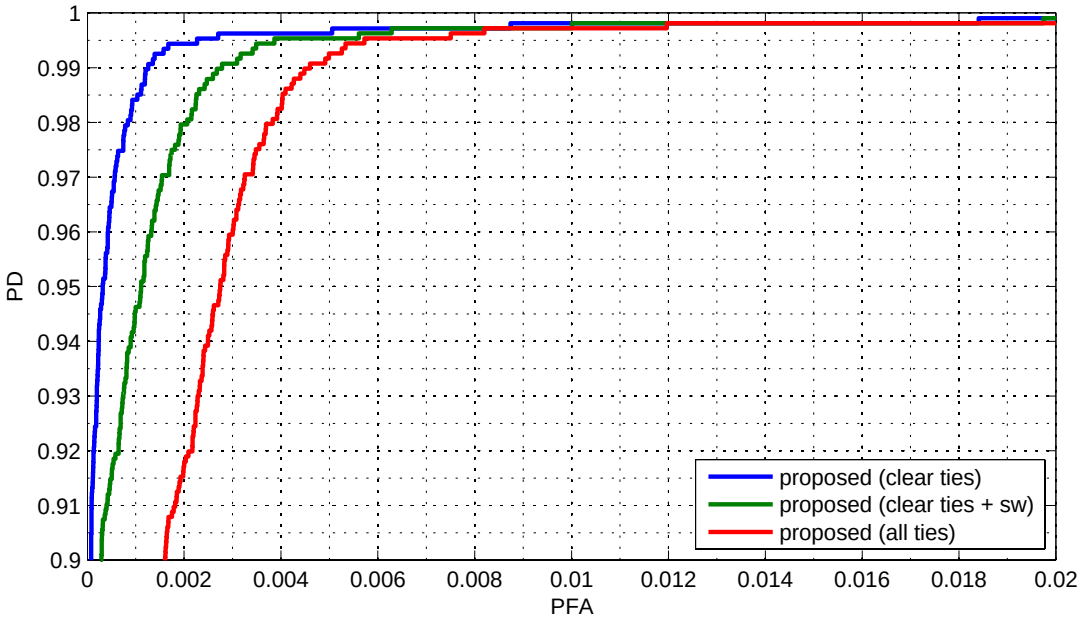


Figure 16. ROC curves for the task of detecting defective (missing or broken) fasteners on the 85-mile testing set.

We used the ROC on clear ties (blue curve) in Figure 16 to determine the optimal threshold to achieve a design false alarm rate of 0.1% ($\tau = 0.1614$). Using this sensitivity level, we ran our defective fastener detector at the tie level (by taking the minimum score across all four regions). Results are shown in Table 9.

Table 9. Results for detection of ties with at least one defective fastener.

Subset	Total ties	Defective	PD	PFA
Clear ties	200,763	1,037	98.36%	0.38%
Clear + switch	201,856	1,045	97.99%	0.71%
All ties	203,287	1,051	98.00%	1.23%

Our evaluation protocol has been to mark the whole tie as "uninspectable" if at least one of the fasteners is not visible in the image. This is not ideal as there are situations where parts of the tie are still "inspectable", for example when the field side of the rail is covered with ballast, but the gage side is inspectable (this explains the six additional defective ties when including uninspectable ties).

6. Crumbling and Chipped Ties Detection

The condition of railway tracks needs to be periodically monitored to ensure safety. Cameras mounted on a moving vehicle such as a hi-rail vehicle or a geometry inspection car can generate large volumes of high resolution images. Extracting accurate information from those images is challenging due to the clutter in the railroad environment. In this section, we describe a novel approach to visual track inspection that uses Deep Convolutional Neural Networks (DCNN) to perform material classification and semantic segmentation on ties. We show that DCNNs that are trained end-to-end for material classification are more accurate than shallow learning machines with hand-engineered features and are more robust to noise. Our approach results in a material classification accuracy of 93.35% using 10 classes of materials. This allows for the detection of crumbling and chipped tie conditions at detection rates of 86.06% and 92.11%, respectively, at a false positive rate of 10 FP/mile on the 85-mile subset introduced in section 1.4.1.

6.1 Background

Texture segmentation and material classification are essential components in any vision-based track inspection system. The crack detection and defective fastener detection results reported in sections 4 and 5 assume that the tie boundary location is available. For example, for a crack detection algorithm to work in practice, it must be paired with a reliable boundary detection algorithm. Indeed, if information is available about the type of material that the tie is made of and the tie region that is covered in ballast, the false alarm rates of both crack and fastener inspection would be drastically reduced. In this section, we describe our method for material classification and semantic segmentation, which can inspect ballast and ties of different materials and detect chipping and crumbling on concrete ties.

As we will see later, the use of predefined texture features followed by a classifier, is not enough for solving the fine-graned material classification problem posed in this section. For example, the system from (Resendiz, Hart, & Ahuja, 2013) is capable of segmenting wood ties from ballast using a combination of Gabor filters and an SVM classifier. However, classifying wood-vs-ballast is a much easier problem than the 10-class problem at hand and requires a more carefully designed approach. Given the vast amount of training data available in our dataset, it is reasonable to resort to deep learning techniques.

The idea of enforcing translation invariance in neural networks via weight sharing goes back to the Neocognitron (Fukushima, 1980). Based on this idea, LeCun *et al.* developed the concept into Deep Convolutional Neural Networks (DCNN) and used it for digit recognition (LeCun, et al., 1989), and later for more general optical character recognition (OCR) (LeCun, Bottou, Bengio, & Haffner, 1998). During the last two years, DCNNs have become ubiquitous in achieving state-of-the-art results in image classification (Krizhevsky, Sutskever, & Hinton, 2013 and C.Szegedy, et al., 2014) and object detection (R.Girshick, J.Donahue, T.Darrell, & J.Malik, 2014). This resurgence of DCNNs has been facilitated by the availability of efficient GPU implementations. More recently, DCNNs have been used for semantic image segmentation. For example, the work of Long, Shelhamer, & Darrell (2014) shows how a DCNN can be converted in to a Fully Convolutional Network (FCN) by replacing fully-connected layers with convolutional ones.

6.2 Approach

In this section we describe the proposed approach to track inspection using material classification and semantic segmentation.

6.2.1 Architecture

Our implementation uses a Fully Convolutional Network (Long, Shelhamer, & Darrell, 2014) based on BVLC Caffe (Jia, et al., 2014). We have a total of 4 convolutional layers between the input and the output layer. The network uses rectified linear units (ReLU) as non-linearity activation functions, overlapping max pooling units of size 3 × 3 and stride of 2. In our experiments we found that dropout is not necessary. Since no preprocessing is done in the sensor, we first apply global gain normalization on the raw image to reduce the intensity variation across the image. This gain is calculated by smoothing the signal envelope estimated using a median filter. We estimate the signal envelope by low-pass filtering the image with a Gaussian kernel. Although DCNNs are robust to illumination changes, normalizing the image to make the signal dynamic range more uniform improves accuracy and convergence speed. We also subtract the mean intensity value that is calculated on the whole training set.

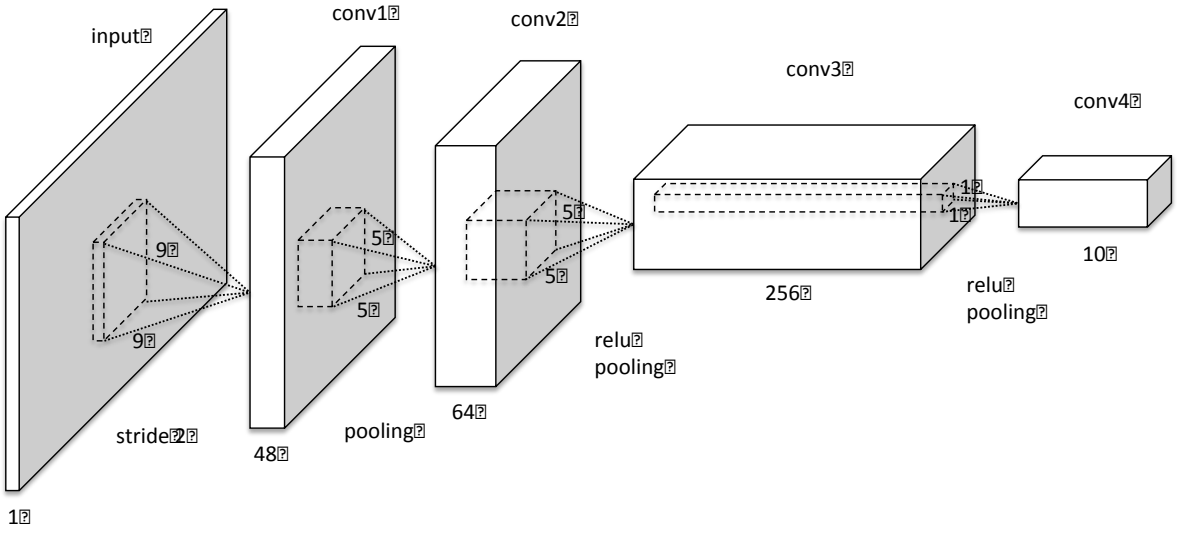


Figure 17**. Network architecture.**

This preprocessed image is the input for our network. The architecture is illustrated in Figure **17**. The first layer takes a globally normalized image and filters it with 48 filters of size 9 × 9. The second convolutional layer takes the (pooled) output of the first layer and filters it with 64 kernels of size 5 × 5 × 48. The third layer takes the (rectified, pooled) output of the second layer and filters it with 256 kernels of size 5 × 5 × 48. The forth convolutional layer takes the (rectified, pooled) output of the third layer and filters it with 10 kernels of size 1 × 1 × 256.

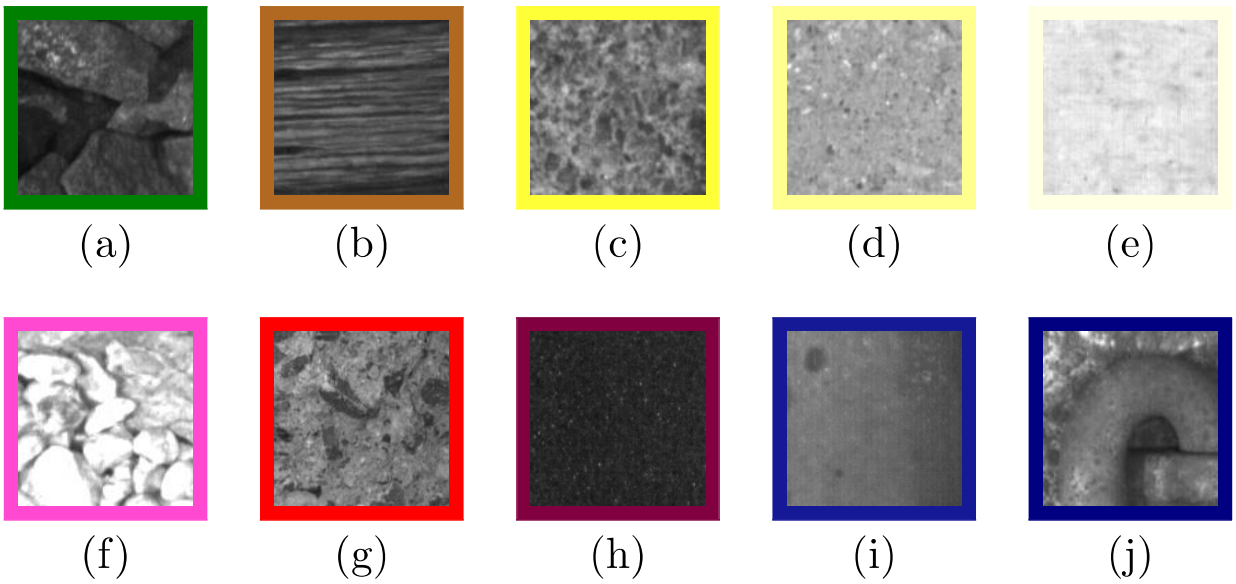


Figure 18. Material categories.

(a) ballast (b) wood (c) rough concrete (d) medium concrete (e) smooth concrete (f) crumbling concrete (g) chipped concrete (h) lubricator (i) rail (j) fastener

The output of the network contains ten score maps at 1/16th of the original resolution. Each value $\Phi_i(x, y)$ in the score map corresponds to the likelihood that pixel location (x, y) contains material of class i. The 10 classes of materials are defined in Figure 18. The network has a total of 493,226 learnable parameters (including weights and biases), of which 0.8% correspond to the first layer, 15.6% to the second layer, 83.1% to the third layer, and the remaining 0.5% to the output layer.

6.2.2 Data Annotation

The ground truth data has been annotated using the annotation tool integrated in the Vision Client described in section 3. The tool allows assigning a material category to each tie as well as its bounding box. The tool also allows defining polygons enclosing regions containing crumbling, chips or ballast. We used the output of our fastener detection algorithm described in section 5 to extract fastener examples.

6.2.3 Training

We trained the network using a stochastic gradient descent on mini-batches of 64 image patches of size 75 × 75. We did data augmentation by randomly mirroring vertically and/or horizontally the training samples. The patches are cropped randomly among all regions that contain the texture of interest. To promote robustness against adverse environment conditions, such as rain, grease or mud, we identified images containing such difficult cases and automatically resampled the data so that at least 50% of the data is sampled from such difficult images.

6.2.4 Score Calculation

To detect whether an image contains a broken tie, we first calculate the scores at each site as

$$S_b(x,y) = \max_{i \notin \mathcal{B}} \Phi_i(x,y) - \Phi_b(x,y)$$

where $b \in \mathcal{B}$ is a defect class (crumbling or chip). Then we calculate the score for the whole image as

$$S_b = \frac{1}{\beta - \alpha} \int_{\alpha}^{\beta} \hat{F}^{-1}(t) dt$$

where $\hat{F}^{-1}(t)$ refers to the *t* sample quantile calculated from all scores $S_b(x,y)$ in the image. The detector reports an alarm if $S > \tau$, where τ is the detection threshold. We used $\alpha = 0.9$ and $\beta = 1$.

6.3 Experimental Results

We evaluated this approach on the 85-mile subset described in section 1.4.1. As we did in the previous section, we downsampled the images by a factor of 2, for a pixel size of 0.86 mm. For the experiments reported in this section, we included all the ties in this section of track, including 140 wood ties that were excluded from the experiments in section 5.2.

6.3.1 Material Identification

We divided the dataset into five splits and used 80% of the images for training and 20% for testing and we generated a model for each of the five possible training sets. For each split of the data, we randomly sampled 50,000 patches of each class. Therefore, for each model was trained with two million patches. We trained the network using a batch size of 64 for a total of 300,000 iterations with a momentum of 0.9 and a weight decay of 0.00005. The learning rate is initially set to 0.01 and it decays by a factor of 0.5 every 30,000 iterations.

In addition to the method described in section 6.2, we have evaluated the classification performance using the following methods:

- **LBP-HF with approximate Nearest Neighbor**: The Local Binary Pattern Histogram Fourier descriptor introduced in (Ahonen, Matas, He, & Pietikäinen, 2009) is invariant to global image rotations while preserving local information. We used the implementation provided by the authors. To perform approximate nearest neighbor we used FLANN (Muja & Lowe, 2009) with the 'autotune' parameter set to a target precision of 70%.
- **Uniform LBP with approximate Nearest Neighbor**: The $LBP_{8,1}^{u2}$ descriptor (Ojala, Pietikäinen, & Mäenpää, 2002) with FLANN.
- **Gabor features with approximate Nearest Neighbor**: We filtered each image with a filter bank of 40 filters (five scales and eight orientations) designed using the code from Haghighat, Zonouz, & Abdel-Mottaleb (2013). As proposed in Manjunath & Ma (1996) we compute the mean and standard deviation of the output of each filter and build a feature descriptor as $f = [\mu_{00}\ \sigma_{00}\ \mu_{01}\ ...\ \mu_{47}\ \sigma_{47}]$. Then, we perform approximate nearest neighbor using FLANN with the same parameters.

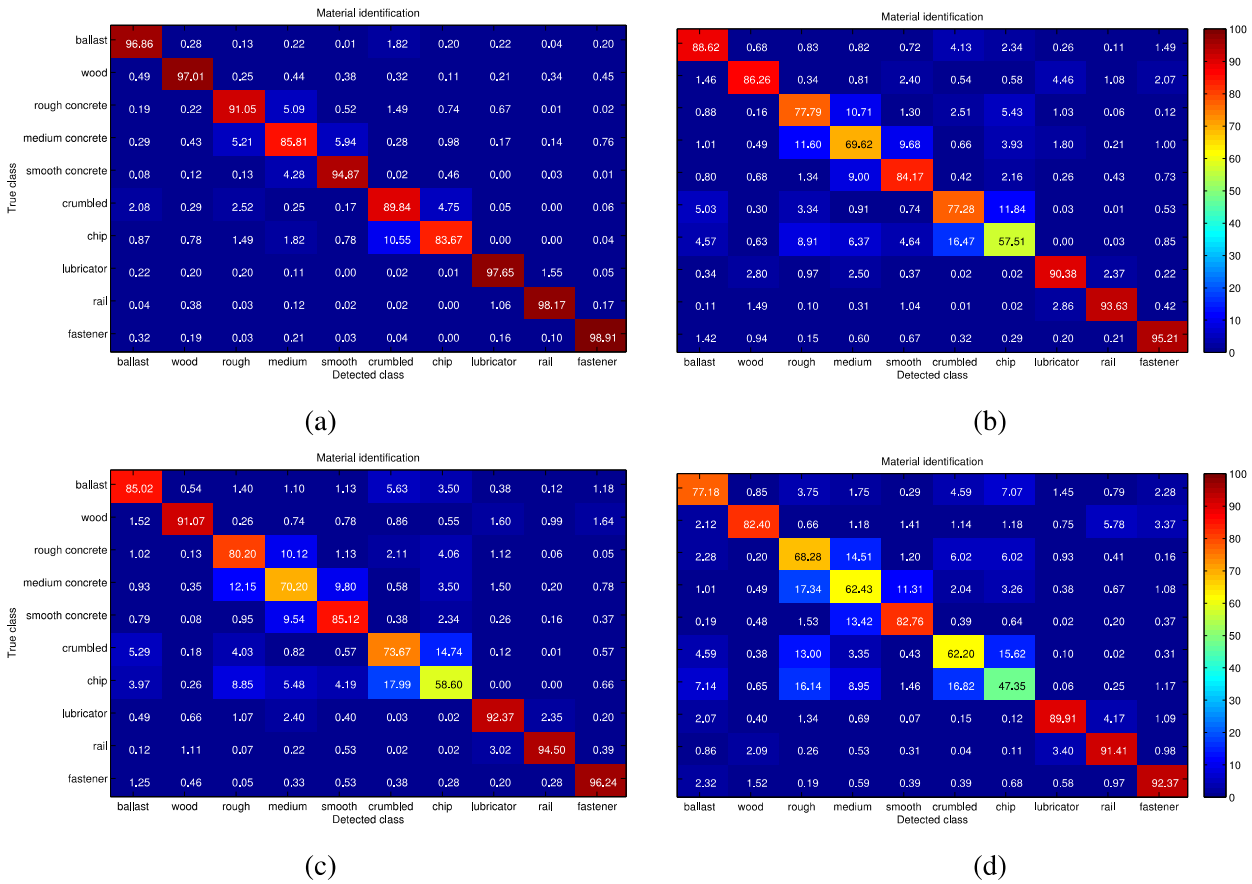


Figure 19. Confusion matrix of material classification on 2.5 million 80 × 80 image patches with (a) Deep Convolutional Neural Networks, (b) LBP-HF with FLANN (c) $LBP_{8,1}^{u2}$ with FLANN (d) Gabor with FLANN.

Table 10. Material classification results.

Method	Accuracy
Deep CNN	**93.55%**
LBP-HF with FLANN	82.05%
$LBP_{8,1}^{u2}$ with FLANN	82.70%
Gabor with FLANN	75.63%

The material classification results are summarized in Table 10 and the confusion matrices in Figure 19.

6.3.2 Semantic Segmentation

Since we are using a fully convolutional DCNN, we directly transfer the parameters learned using small patches to a network that takes one 4096 × 320 image as an input, and generates 10 score maps of dimension 252 × 16 each. The segmentation map is generated by taking the label corresponding to the maximum score. Figure 7 shows several examples of concrete and wood ties, with and without defects and their corresponding segmentation maps.

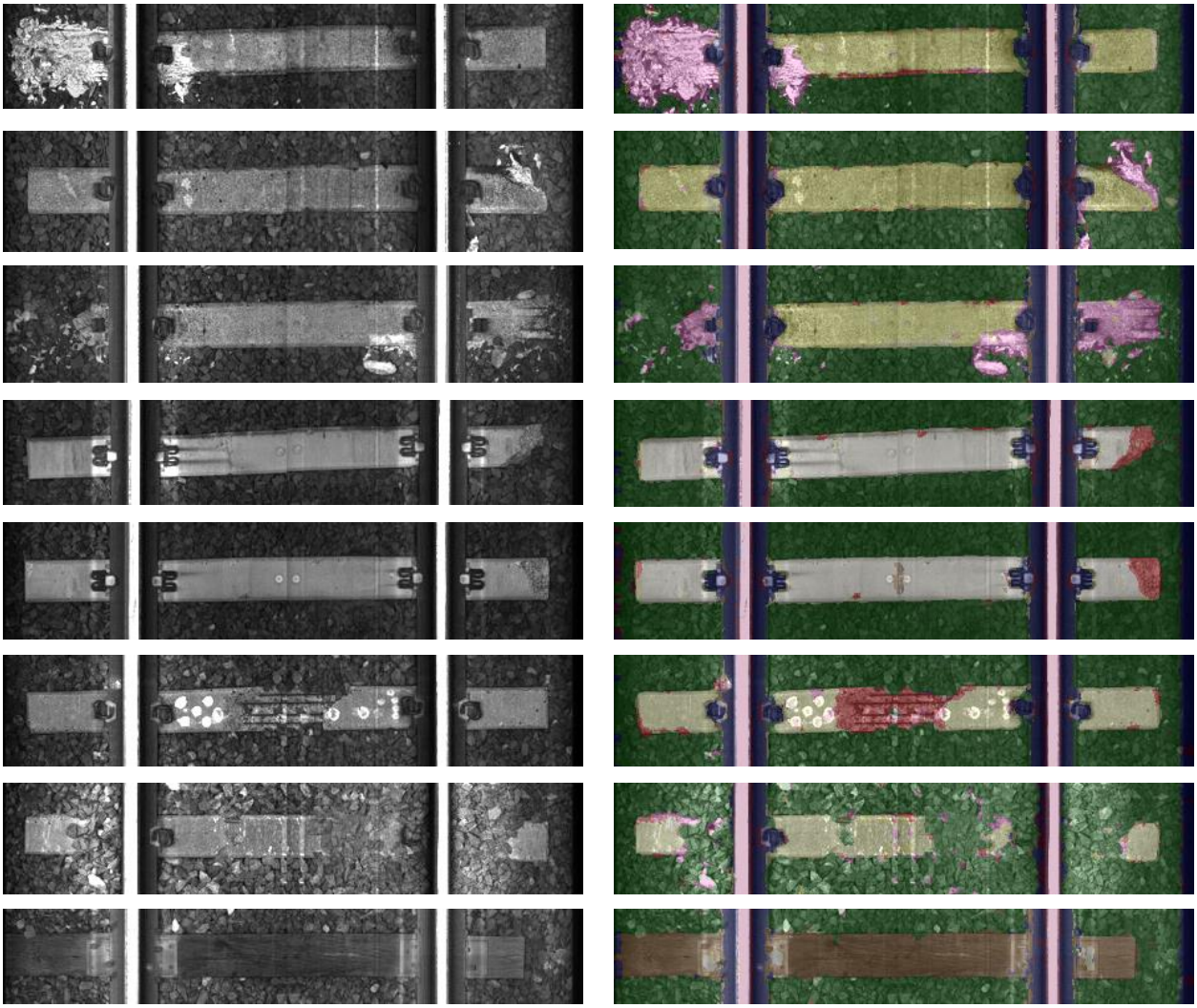


Figure 20. Semantic segmentation results.

Note 1: Images displayed a 1/16 of original resolution.

Note 2: See Figure 18 for color legend.

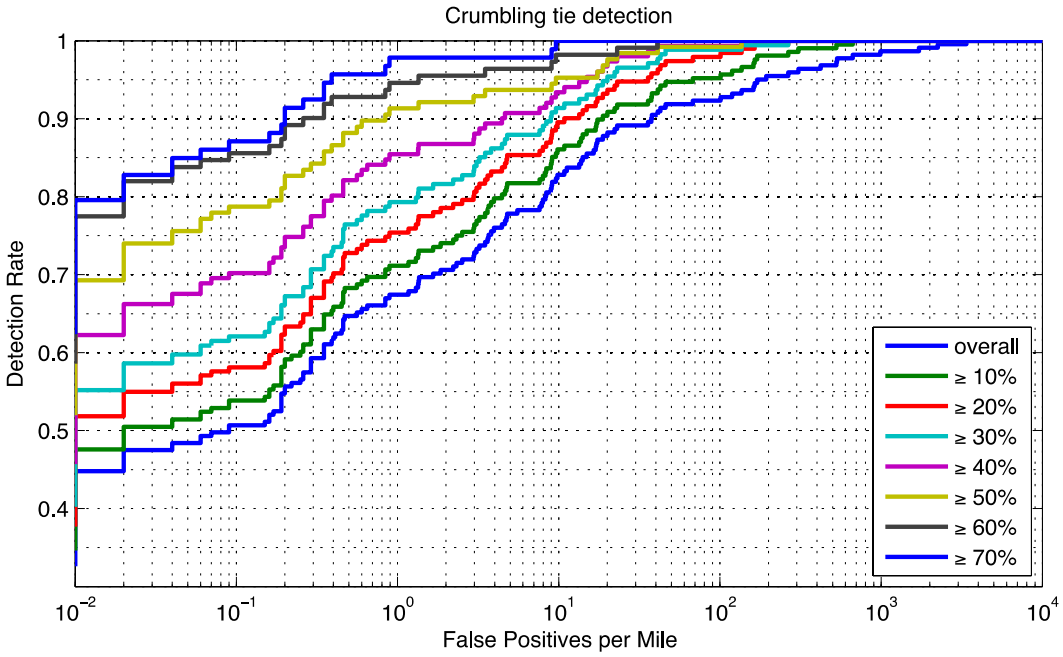


Figure 21. ROC curve for detecting crumbling tie conditions

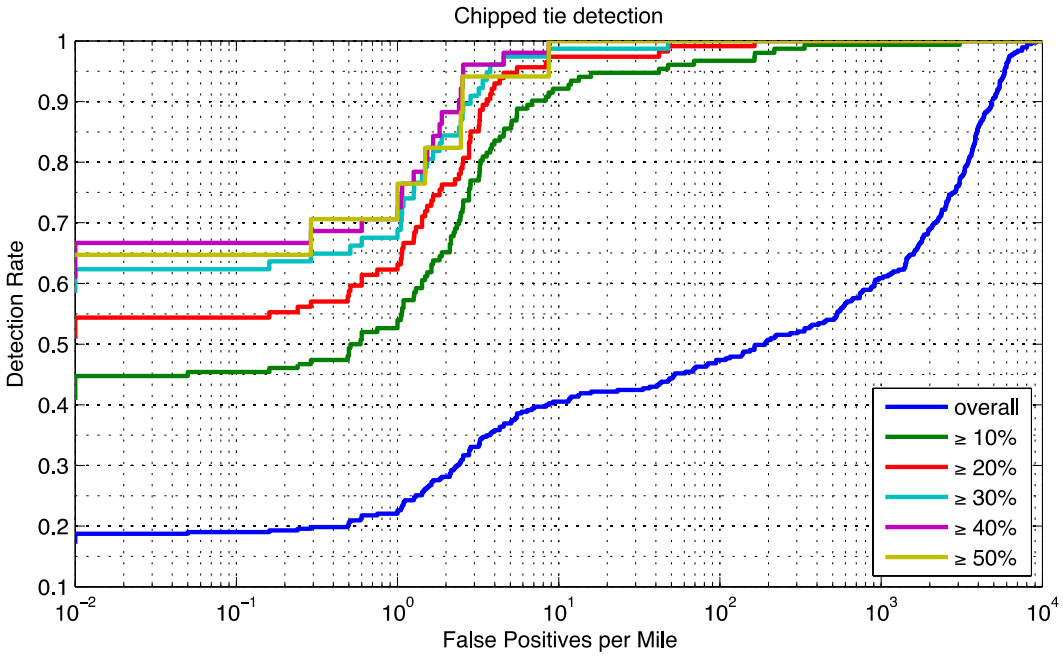


Figure 22. ROC curve for detecting chip tie conditions

Note 1: Each curve is generated considering conditions at or above a certain severity level.

Note 2: False positive rates are estimated assuming an average of 10^{-4} images per mile. Confusions between chipped and crumbling defects are not counted as false positives.

6.3.3 Crumbling and Chipped Tie Detection

The first three rows in Figure 20 show examples of crumbling ties and their corresponding segmentation map. Similarly, rows 4 through 6 show examples of chipped ties. To evaluate the accuracy of the crumbling and chipped tie detector described in Section 6.2.4 we divide each tie into four images and we evaluate the score on each image independently. Due to the large variation in the area affected by crumbling/chip we assigned a severity level to each ground truth defect, and for each severity level we plot the ROC curve of finding a defect when ignoring lower level defects. The severity levels are defined as the ratio of the inspectable area to the area that is labeled as a defect. Figure 21 shows the ROC curves for crumbling tie detection at each severity level. Similarly, Figure 22 shows ROC curves for chipped tie detection. Because the fixed $\alpha = 0.9$ in section 6.2.4 is chosen, the performance is not reliable for defects under 10% severity. For defects that are bigger than the 10% threshold, at a false positive rate of 10 FP/mile the detection rates are 86.06% for crumbling and 92.11% for chips.

7. Conclusions and Future Work

7.1 Industry Feedback

During the execution of this project, we held four meetings and demonstrations with our industry partners: Amtrak and ENSCO, Inc. Our partners evaluated our Vision Client application, and provided feedback. For example, several bugs and usability shortcomings were identified and corrected early on. This feedback prevented such mistakes from degrading the quality of the overall research. Early into the project, we decided to integrate manual tie grading and tie alignment functionality into the Vision Client and our backend, so ENSCO reviewers could complete the annotation tasks needed for both the tie degradation study and the anomaly detection project with the same interface. This allowed the reviewers to learn a single tool and resulted in higher productivity, facilitating the creation of the largest annotated computer vision dataset for railway track inspection that we are aware of.

The client-server architecture has permitted rapid deployment of software updates. For example, during the last two years, we have released a total of 71 versions, allowing changes to be tested quickly. This project used agile development practices, which has led to a very stable codebase with zero outstanding critical bugs. The feedback from the industry has been positive and has helped us prioritize our research and development efforts.

7.2 General Software Development Roadmap

The software has grown into a codebase of more than 46K source lines of code (SLOC) in an orderly fashion. The modular design has allowed us to encapsulate different functions with a limited number of dependencies and maximum code reuse. However, as we transfer this technology to the industry and add developers from different institutions to the project, some reorganization and code refactoring may be necessary. For example, it may be beneficial to migrate our existing messaging and logging functions from our hand-coded approach into Google's *protobuf* and *glog*, as these libraries are both open-source under a BSD license and have a large user base. It may also be beneficial to improve code documentation, so new developers can learn quickly and contribute new functionality to the project.

At the beginning of the project, a great deal of care was taken to make sure that we have a design that can scale up for future needs. As the project keeps growing in size, number of contributors, and user base, it may be beneficial to reevaluate the design to make sure that the framework does not have any bottlenecks that could hamper future innovation.

7.3 Crack Detection

Crack detection was the first component that we worked on and it has been the most challenging part of this project. Due to time and budget limitations, we had to address fastener inspection and crumbling/chip detection problems before we could deploy a robust crack detection solution. Although we have basic components for extracting cracks regions and generating crack skeletons, they are not tuned and optimized for large-scale deployment. However, the experience gained from developing the crack detection module has helped us design and implement the other two anomaly detection modules. Moreover, crack detection is not possible in isolation, as contextual information (provided by the other two modules) is necessary in order to avoid false alarms due to tie edges, fastener edges, ballast edges and so on. Therefore, as new funding for

crack detection becomes available, we believe that a complete crack detection module can be developed in a short period time.

7.4 Fastener Detection

In order to advance the state-of-the-art in automated railway fastener inspection, our design projects the samples into a representation that minimizes intra-class variation while maximizing inter-class separation. To achieve minimum intra-class variation, we use the HOG features (which have built-in intensity normalization) while preserving the distinctive distribution of edges. Our sophisticated graphical user interface facilitates accurate alignment of the fastener locations to avoid intraclass variations due to misalignment. To achieve maximum inter-class separation while maintaining the principle of parsimony, we resort to the maximum margin formulation and simplicity offered by linear SVMs. We enforce intra-class separation during the sampling of the training data. For the fastener-vs-background classifiers, we bootstrapped difficult samples when we built the training set and for the fastener-vs-rest classifiers, we aligned the negative samples to the most confusing position, so the learning machine could focus on the best way to separate classes on the most distinctive parts of the object.

The detector discussed in section 5 is based on inner products between feature vectors that were extracted from image patches and a set of templates. Therefore, the computation cost is the cost of calculating the feature vector plus performing the inner products with each the two template vectors of each class. We have chosen the HOG as our feature vector, but other (probably simpler) alternative representations are possible and they may dramatically speed-up the computation time without significantly degrading the detection performance. Alternatively, we could speed-up the computation of the inner products by reducing the dimensionality of the feature vector by using Principal Component Analysis (PCA).

Although the approach described here works most of the time and can handle a wide variety of track conditions, including mud splashes, heavy grease, and partial occlusions due to small branches, leaves, pieces of ballast or cables, there is still room for improvement. Indeed, due to the requirement of using fully annotated training samples our approach is statistically inefficient. In the future we plan to extend the training algorithm to allow it to learn from weakly labeled data. For weakly label data, we refer to the situation where we know that all fasteners in a range of ties are in good condition, but we do not know the exact location and type of each individual fastener.

Also, the decision is currently based on an image-by-image basis and disregards the statistical dependencies of fastener location as well as fastener type between adjacent ties. Adding such dependencies through a Markov model would probably reduce spurious detection and classification errors. Moreover, specific models for the arrangement of fasteners around switches and other special track structures could be used to reduce the uncertainty in fastener detection that our solution has under such scenarios. In the future, we plan to address some of these issues and extend this framework by adding more types of fasteners and using more robust matching methods. Nevertheless, we believe that the system described here is a big step towards automated visual track inspection and can be used by the railroad industry in production mode.

7.5 Crumbling and Chipped Tie Detection

Using the proposed fully-convolutional deep CNN architecture we have shown that it is possible to accurately localize and inspect the condition of railway components using grayscale images.

We believe that our method performs better than traditional texture features because DCNNs can capture more complex patterns and reuse patterns learned with increasing levels of abstraction that are shared among all classes. This explains why there is much less overfitting on the anomalous classes (crumbled and chip) despite having a relatively limited amount of training data.

We currently run the network in a feed-forward fashion. In the future, we plan to continue exploring recursive architectures to discover long-range dependencies among image regions with the purpose of better separate normal regions from anomalous ones. Also, as previously discussed, accuracy can be improved by adding the ability to learn from ambiguously labeled data.

7.6 Automation and Deployment

The current version of the software runs in batch mode and takes one mile of data at a time. For a successful deployment, it will be necessary to modify the software so it can run in streaming mode. Also, once the target computing platform is selected, it will be necessary to tune the algorithms so they utilize the computing resources efficiently.

Amtrak has indicated that their first deployment will be on a hi-rail vehicle. Although hi-rail platforms have the advantage of lower processing speed requirements, the limitations of rack space, ventilation and electric power also pose challenges. We plan to work closely with Amtrak to ensure that their system design is compatible with the architecture described in this report.

7.7 Future Research Topics

Not only will we improve and refine existing algorithms, we plan to investigate the following topics:

- **Learning with weakly labeled anomalies**: Existing detectors require that humans label each training sample with the exact fastener type and draw a bounding box around each component. We plan to develop learning algorithms that only need to know whether or not a tie contains an anomaly of a specific type.
- **Domain adaptation for anomaly detection under changes in operating conditions**: The performance of existing methods significantly degrades when operating conditions change. For example, when the training set contains clear dry ties and the testing set contains ties covered with grease, the detection performance will degrade. Research in unsupervised domain adaptation algorithms that can handle the presence of anomalies in the target domain will allow existing and new algorithms to produce more accurate results.
- **Extreme value theory for adaptive anomaly detection**: Existing detectors generate bursts of false alarms when the signal to noise ratio degrades. In order to keep a constant false alarm rate, it is necessary to use an adaptive threshold. Extreme value theory provides a theoretical framework for estimating the density at the upper tail of a probability distribution, which can be used for adaptive thresholding. We plan to develop adaptive threshold estimation algorithms that can handle the presence of anomalies.
- **Automated tie grading:** Railroads rely on numerical tie grades to plan for their track maintenance. Using the results of existing algorithms, research methods for prediction of

numerical tie grades based on rules provided by the user as well as learned from examples.

- **Track component detection**: Find the bounding rectangle for each track component (rails, fasteners, ties, bolts, switch points). Estimate parameters derived from component detections, such as intertie distances, base gage, and distinctive bolts and fastener patterns to help with tie matching.
- **Tie matching**: For each detected tie, automatically find the corresponding tie on previous surveys. It is assumed that approximate GPS location and/or milepost is available, but the direction of travel and track number may be unknown.
- **Tie alignment**: For each pair of matched ties find a number of corresponding points and warp the target image so it aligns with the source image. Tie alignment shall be robust to local changes and noise.
- **Change detection and characterization**: Detect local differences (after intensity-normalization and denoising) between aligned images. Classify each detected change into relevant or not relevant, using previously trained relevancy criteria and patterns of false changes such as debris, grease, mud, leaves, water, snow. Differences will be summarized as to whether a specific component (tie or clip) has been replaced, the component has developed new damage, or that previously detected damage has worsened.
- **Depth from stereo**: Conduct research into algorithms for reconstructing depth from two or more cameras in the railway environment and examine methods for extracting information from such point clouds, such as estimating potential rail seat abrasion conditions or changes in ballast height that can disambiguate whether ballast is covering a defective tie or a good one.
- **Color imaging**: Research algorithms to find new types of anomalies based on color, such as corrosion, vegetation, or mud pumping conditions.

7.8 Conclusion

This report has described a new approach for inspecting railway tracks using recent advances in the area of computer vision and pattern recognition. The algorithms described in this report have been packaged into an integrated software suite that will allow different railroad users to reconfigure it for their specific needs. We believe that the University-Industry partnership that has been forged during this project will continue in future years. The number of problems in the railway industry that could be solved with computer vision and pattern recognition techniques is large, and the work described in this report is just a tiny fraction of what can be done. We hope that in the future, the railway industry will consider releasing other datasets to the research community, so progress towards other safety-related problems can be made.

8. References

Ahonen, T., Matas, J., He, C., & Pietikäinen, M. (2009). Rotation invariant image description with local binary pattern histogram Fourier features. *Image Analysis*, 61-70.

Babenko, P. (2009). *Visual inspection of railroad tracks.* University of Central Florida. PhD thesis.

Berry, A., Nejikovsky, B., Gibert, X., & Tajaddini, A. (2008). High speed video inspection of joint bars using advanced image collection and processing. *Proc. of World Congress on Railway Research.*

Bobin, J., Starck, J.-L., Fadili, M., Moudden, Y., & Donoho, D. (2007). Morphological component analysis: an adaptive thresholding strategy. *IEEE Transactions on Image Processing, 16*(11), 2675-2681.

C.Szegedy, W.Liu, Y.Jia, P.Sermanet, S.Reed, D.Anguelov, et al. (2014). Going deeper with convolutions. *arXiv:1409.4842* .

Canny, J. (1986). A computational approach to edge detection. *IEEE Trans. Pattern Analysis and Machine Intelligence, 8*(6), 679-698.

Chambon, S., & Moliard, J. (2011). Automatic road pavement assessment with image processing: Review and comparison. *Int. Journal of Geophysics*, doi:10.1155/2011/989354.

Cunha, A., Zhou, J., & Do, M. (2006). The nonsubsampled contourlet transform: Theory, design, and applications. *IEEE Transactions on Image Processing, 15*(10), 3089-3101.

Cunningham, J., Shaw, A., & Trosino, M. (2000, May). *Patent No. 6,064,428.* US.

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), 1*, pp. 886-893.

De Ruvo, P., Distante, A., Stella, E., & Marino, F. (2009). A GPU-based vision system for real time detection of fastening elements in railway inspection. *IEEE International Conference on Image Processing (ICIP)*, (pp. 2333-2336).

Easley, G., Labate, D., & Negi, P. (2013). 3D data denoising using combined sparse dictionaries. *Math. Model. Nat. Phenom., 8*(1), 60-74.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics, 36*(4), 93-202.

Gibert, X., Berry, A., Diaz, C., Jordan, W., Nejikovsky, B., & Tajaddini, A. (2007). A machine vision system for automated joint bar inspection from a moving rail vehicle. *ASME/IEEE Joint Rail Conference & Internal Combustion Engine Spring Technical Conference.*

Haghighat, M., Zonouz, S., & Abdel-Mottaleb, M. (2013). Identification using encrypted biometrics. *Computer Analysis of Images and Patterns*, 440-448.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., et al. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2013). Imagenet classification with deep convolutional neural networks. *NIPS*.

Kutyniok, G., & Lim, W. (2011). Image separation using wavelets and shearlets. *In: Curves and Surfaces (Avignon, France, 2010), Lecture Notes in Computer Science 6920*.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., et al. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation, 1*(4), 541-551.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998, November). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*.

Li, Y., Trinh, H., Haas, N., Otto, C., & Pankanti, S. (2014, April). Rail component detection, optimization, and assessment for automatic rail track inspection . *IEEE Trans. on Intelligent Transportation Systems, 15*(2), 760-770.

Long, J., Shelhamer, E., & Darrell, T. (2014). Fully convolutional networks for semantic segmentation. *arXiv:1411.4038*.

Ma, C., Zhao, C., & Hou, Y. (2008). Pavement distress detection based on nonsubsampled contourlet transform. *Int. Conf. on Computer Science and Software Engineering*, *1*, pp. 28-31.

Mahalanobis, A., Kumar, B. V., Song, S., Sims, S. R., & Epperson, J. F. (1994, June). Unconstrained correlation filters. *Appl. Opt., 33*(17), 3751-3759.

Manjunath, B., & Ma, W. (1996). Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 18*(8), 837-842.

Marino, F., Distante, A., Mazzeo, P. L., & Stella, E. (2007). A real-time visual inspection system for railway maintenance: automatic hexagonal-headed bolts detection. *IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 37*(3), 418-428.

Muja, M., & Lowe, D. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. *International Conference on Computer Vision Theory and Application (VISSAPP'09)* (pp. 331-340). INSTICC Press.

Ojala, T., Pietikäinen, M., & Mäenpää, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24*(7), 971-987.

Oliveira, H., & Correia, P. (2013). Automatic road crack detection and characterization. *IEEE Transactions on Intelligent Transportation Systems, 14*(1), 155-168.

Platt, J. C., Cristianini, N., & Shawe-taylor, J. (1999). Large margin DAGs for multiclass classification. *NIPS, 12*, pp. 547-553.

R.Girshick, J.Donahue, T.Darrell, & J.Malik. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *IEEE Computer Society Conference in Computer Vision and Pattern Recognition (CVPR).*

Resendiz, E., Hart, J., & Ahuja, N. (2013, June). Automated visual inspection of railroad tracks. *IEEE Trans. on Intelligent Transportation Systems, 14*(2), 751-760.

Sahu, S., & Thaulow, N. (2004). Delayed ettringite formation in swedish concrete railroad ties. *Cement and Concrete Research, 34*, 1675-1681.

Shehata, M. H., & Thomas, M. D. (2000). The effect of fly ash composition on the expansion of concrete due to alkalisilica reaction. *Cement and Concrete Research, 30*, 1063-1072.

Smak, J. A. (2012). Evolution of Amtrak's concrete crosstie and fastening system program. *International Concrete Crosstie and Fastening Symposium.*

Starck, J.-L., Elad, M., & Donoho, D. (2005). Image decomposition via the combination of sparse representation and a variational approach. *IEEE Transactions on Image Processing, 14*(10), 1570-1582.

Subirats, P., Dumoulin, J., Legeay, V., & Barba, D. (2006). Automation of pavement surface crack detection using the continuous wavelet transform. *IEEE International Conference on Image Processing*, (pp. 3037-3040).

Trinh, H., Haas, N., Li, Y., Otto, C., & Pankanti, S. (2012). Enhanced rail component detection and consolidation for rail track inspection. *IEEE Workshop on Applications of Computer Vision (WACV)*, 289-295.

Trosino, M., Cunningham, J., & Shaw, A. (2002, March). *Patent No. 6,356,299.* US.

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), 1*, 511-518.

Abbreviations and Acronyms

ASR	Alkali-Silicone Reaction
AUC	Area Under the Curve
BAA	Broad Agency Announcement
BSD	Berkeley Software Distribution
BVLC	Berkeley Vision and Learning Center
CFR	Code of Federal Regulation
CMOS	Complementary Metal-Oxide Semiconductor
CNN	Convolutional Neural Network
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CTIV	Comprehensive Track Inspection Vehicle
CUDA	Common Unified Device Architecture
DCNN	Deep Convolutional Neural Network
DAG	Directed Acyclic Graph
DEF	Delayed Ettringite Formation
DST	Discrete Shearlet Transform
DWT	Discrete Wavelet Transform
FCN	Fully-Convolutional Network
FFT	Fast Fourier Transform
FLANN	Fast Library for Approximate Nearest Neighbors
FP	False Positives
GPGPU	General-Purpose Graphics Processing Unit
GPL	General Public License
GPU	Graphics Processing Unit
GPS	Global Positioning System
HOG	Histogram of Oriented Gradients
HSR	High Speed Rail
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP over SSL
IPP	Integrated Performance Primitives

LBP	Local Binary Patterns
LBP-HF	Local Binary Patterns Histogram Fourier
LED	Light Emitting Diode
LGPL	Lesser General Public License
NEC	Northeast Corridor
MUSIC	MUltiple SIgnal Classification
OS	Operating System
OT-MACH	Optimal Trade-off Maximum Average Correlation Height
PD	Probability of Detection
PDA	Personal Digital Assistant
PFA	Probability of False Alarm
RC	Release Candidate
ReLU	Rectified Linear Unit
ROC	Receiver Operating Characteristic
ROI	Region of Interest
SDK	Software Development Kit
SP	Service Pack
SLOC	Source Lines of Code
SSL	Secure Sockets Layer
SVM	Support Vector Machine
UMD	University of Maryland
UMIACS	University of Maryland Institute for Advanced Computer Studies
VPN	Virtual Private Network