HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Semesterarbeit, Abteilung Informatik

Extraction of Crosswalks from Aerial Images

HSR Hochschule für Technik Rapperswil

Herbstsemester 2015
18. Dezember 2015

Autoren: Bühler Severin & Kurath Samuel

Betreuer: Prof. Keller Stefan

Arbeitsperiode: 16.09.2015 - 18.12.2015

0.1 Abstract

Fussgängerstreifen sind ein essentieller Bestandteil der Fussgängernavigation. Sie sind jedoch nur spärlich in OpenStreetMap erfasst, was somit zu nicht optimal geplanten Routen führt.

Um dem entgegen zu wirken, befasst sich dieses Projekt mit der automatischen Erkennung von Fussgängerstreifen auf Orthofotos (Satellitenbildern). Dabei entstand eine hoch skalierbare Applikation, die gelbe Fussgängerstreifen entlang von Strassen findet und ihre Koordinaten extrahiert. Die Erkennung erfolgt durch ein Convolutional Neural Network, welches im Bereich des Deep Learnings angesiedelt ist. Es wurde eine Erkennungsrate von über 80% der visuell sichtbaren Fussgängerstreifen mit einer Fehlerrate von weniger als 10% erreicht. Dieser Prozess konnte mit Hilfe eines Queueing Systems auf eine grosse Anzahl von Computern verteilt werden, was die Verarbeitung so vieler Daten in angemessener Zeit überhaupt erst ermöglicht. Die Koordinaten der Fussgängerstreifen wurden im Anschluss an das Crowdsourcing-System MapRoulette übergeben. Auf diesem Weg konnten die Daten in OpenStreetMap integriert werden und zur Verbesserung der Fussgängernavigation ihren Anteil beitragen.

Es ist möglich diese Lösung so auszubauen, dass der Erkennungsalgorithums auf weitere Objekte angewendet werden kann.

Weitere Informationen: `https://github.com/geometalab/OSM-Crosswalk-Detection`

Crosswalks are an essential part of pedestrian navigation. Unfortunately, they are recorded only sparsely in OpenStreetMap. This leads to non-optimal routes.

To counteract this problem, the topic of this poject is to automate the process of finding crosswalks on orthophotos (satellite images). The result is a highly scalable application which finds yellow crosswalks along streets and extracts their coordinates. The recognition is implemented with a convolutional neural network that is located in the area of deep learning. It has achieved a recognition rate of over 80% with a false discovery rate of less than 10%. This process could be shared on a large number of computer by using a queuing system that makes the processing of so much data possible. After that, the coordinates were added to the crowdsourcing system MapRoulette. With this solution the coordinates of the crosswalks can be integrated in openstreetmap to help improve pedestrian navigation.

It is possible to expand this solution so that the the algorithm can be applied to other objects.

Further informations: `https://github.com/geometalab/OSM-Crosswalk-Detection`

0.2 Management Summary

Ausgangslage

OpenStreetMap (OSM) ist ein Wikipedia-artiges Projekt mit einer Webkarte ähnlich wie Google Maps. OSM bietet unter anderem freie Geodaten, die auch für Fussgängernavigation genutzt werden können. Dabei müssen die bestehenden Fussgängerstreifen in die Routenplanung einbezogen werden, um eine Überquerung der Strassen zu ermöglichen. Gemäss heutigem Stand sind noch lange nicht alle Fussgängerstreifen in OSM erfasst, was die Fussgängernavigation erschwert.

Dieses Projekt ist der Versuch einer automatischen Erkennung von Fussgängerstreifen auf Orthofotos (Satellitenbilder). Da das maschinelle Erfassen von Daten in OSM nicht erlaubt ist, können die gefundenen Koordinaten in ein Crowdsourcing-System wie zum Beispiel MapRoulette eingespeist werden. In diesem bewerten Nutzer, ob die eingefügten Daten korrekt sind oder nicht.

Vorgehen

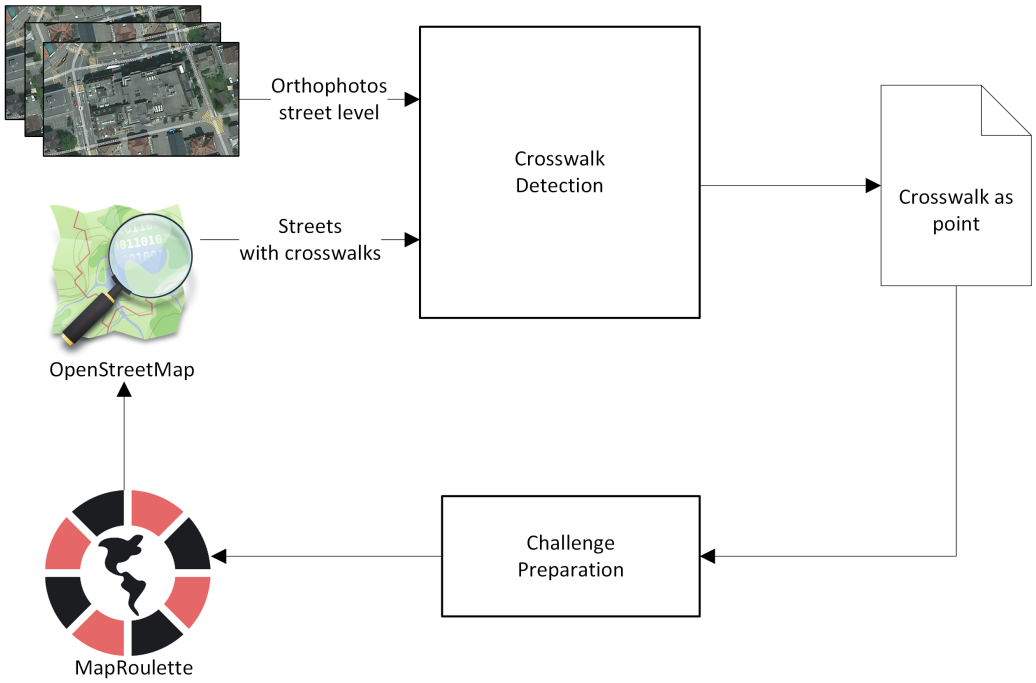

Abbildung 1: Überblick über den Datenfluss zur Erkennung und Erfassung von Fussgängerstreifen in OpenStreetMap.

Die Arbeit begann mit der Evaluation eines passenden Bilderkennungsalgorithmus. Dabei wurden diverse Algorithmen zur Klassifikation der Bilder geprüft, vom Haar Feature-based Cascade Classifier über Fast Fourier Transform bis zu Neuronalen Netzen. Ein neuronales Netz, genauer ein Convolutional Neural Network, brachte die besten Resultate.

Weiter mussten Orthofotos, wie auch Informationen zu Strassenachsen und die Koordinaten der schon in OSM erfassten Fussgängerstreifen beschafft werden. Dabei konnte auf Bing Maps und die OSM-Programmierschnittstelle von MapQuest zurückgegriffen werden.

Da die Bilderkennung ein sehr rechenintensiver Prozess ist, musste der Erkennungsprozess parallelisiert werden. Mit Hilfe eines Docker Images und einem Queueing System konnte die Erkennung auf verschiedene Maschinen verteilt und der Erkennungsaufwand auf mehrere Tage beschränkt wer-

den. Ohne diese Parallelisierung hätten Wartezeiten von mehreren Wochen in Kauf genommen werden müssen.

Ergebnisse

Aus diesem Projekt entstand eine Applikation für die automatische Erkennung von Fussgängerstreifen. Diese bezieht in einem angegebenen Bereich, wie beschrieben, automatisch die entsprechenden Orthofotos und Strasseninformationen und extrahiert daraus die Koordinaten der Fussgängerstreifen. Die Koordinaten werden in einer JSON-Datei abgelegt und können in eine MapRoulette Challenge umgewandelt werden.

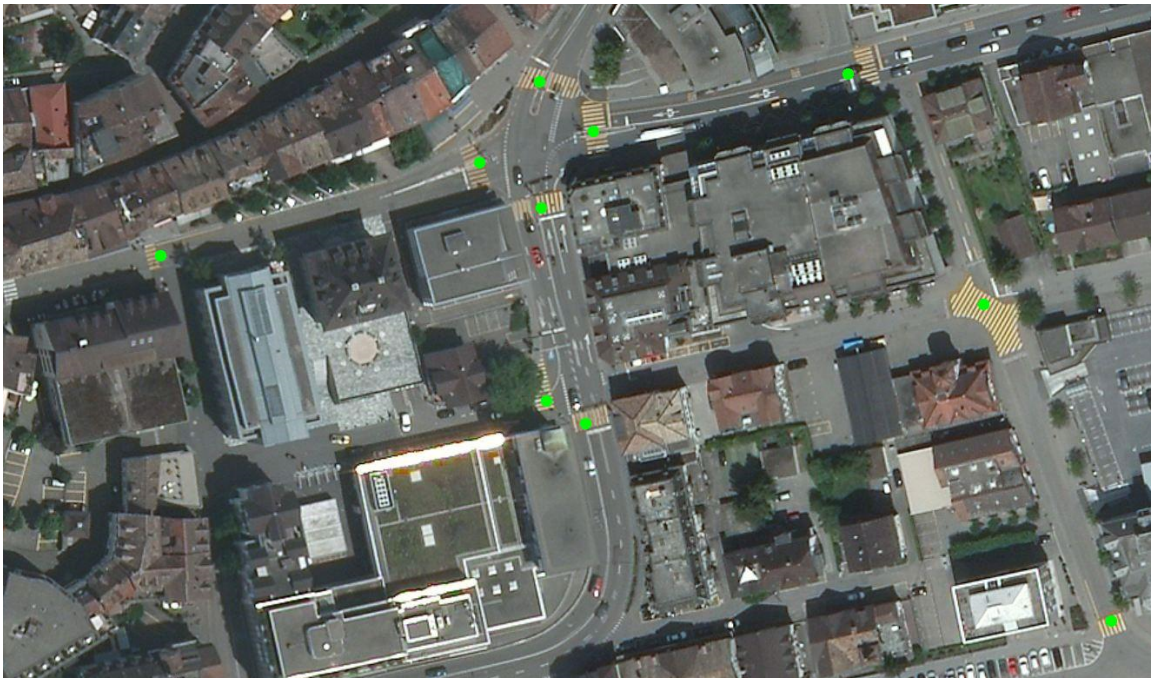

Abbildung 2: Rapperswil Innenstadt - Gefundene Fussgängerstreifen sind mit einem grünen Punkt markiert.

Die Applikation erkannte mehr als 80% aller gelben Fussgängerstreifen mit einer Fehlerrate von weniger als 10%. Die Koordinaten der Fussgängerstreifen im Raum der Kantone Zürich und Zug konnten an MapRoulette zur Einpflege in OSM übergeben werden.

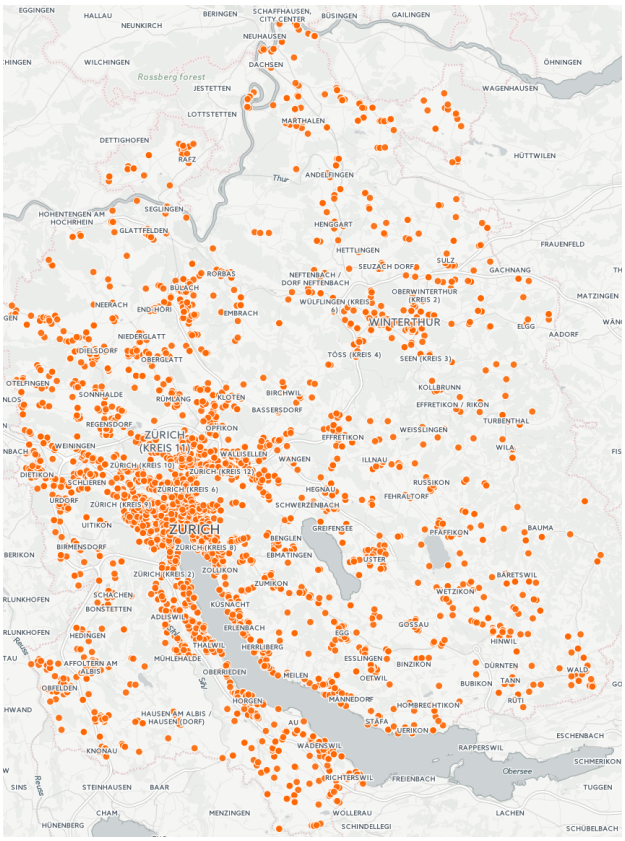

Abbildung 3: Gesamtresultat Kanton Zürich

Ausblick

Der Erkennungsalgorithmus ist zu Ende dieses Projektes auf die Region Zürich und die Ostschweiz spezialisiert. Mit weiteren Optimierungen beim Neuronalen Netz ist es möglich, alle Fussgängerstreifen in der Schweiz zu erfassen und sogar weisse Fussgängerstreifen für andere europäische Länder zu erkennen. Auch weitere Strassenmarkierungen sind denkbar.

Weiter ist die Geschwindigkeit, in der Convolutional Neural Network verbessert werden, enorm. In 2 - 3 Jahren könnte es möglich sein, annähernd alle Fussgängerstreifen auf Orthofotos zu erkennen.

Inhaltsverzeichnis

0.1 Abstract . . . 2
0.2 Management Summary . . . 3
0.3 Aufgabenstellung . . . 12

1 Technischer Bericht **14**
1.1 Einführung . . . 15
1.1.1 Vision . . . 15
1.2 Stand der Technik . . . 16
1.2.1 Literaturrecherche . . . 16
1.3 Evaluation Suchalgorithmus . . . 18
1.3.1 Algorithmen Vergleich . . . 18
1.3.2 Auswertung . . . 21
1.4 Evaluation Crowdsourcing-System . . . 22
1.4.1 MapRoulette . . . 22
1.4.2 To-Fix . . . 23
1.4.3 Auswertung . . . 24
1.5 Umsetzungskonzept . . . 25
1.5.1 OpenStreetMap . . . 25
1.5.2 Bing . . . 26
1.5.3 Convolutional Neural Network . . . 26
1.5.4 Parallelisierung . . . 28
1.5.5 MapRoulette . . . 29
1.6 Resultate . . . 30
1.6.1 Zielerreichung . . . 30
1.6.2 Resultierende Dateien . . . 31
1.6.3 Persönlicher Bericht . . . 33
1.6.4 Dank . . . 34

2 Software Projektdokumentation **35**
2.1 Vision . . . 36
2.2 Anforderungsspezifikation . . . 37
2.2.1 Use Case . . . 37
2.2.2 Sequenzdiagramm . . . 40
2.2.3 Nichtfunktionale Anforderungen . . . 42
2.2.4 Analyse . . . 43
2.3 Implementation . . . 46
2.3.1 Data Layer . . . 46
2.3.2 Detection Layer . . . 51
2.3.3 Role Layer . . . 55
2.4 Test . . . 56

2.4.1 Unittests . . . 56
2.4.2 CircleCI . . . 56

3 Projektmanagement 57
3.1 Rollen und Verantwortlichkeiten . . . 58
3.1.1 Prof. Keller Stefan . . . 58
3.1.2 Bühler Severin . . . 58
3.1.3 Kurath Samuel . . . 59
3.2 Entwicklungsumgebung und Infrastruktur . . . 60
3.2.1 IDE (Integrated Development Environment) . . . 60
3.2.2 SCM (Source Control Management) . . . 60
3.2.3 CI (Continuous Integration) . . . 60
3.2.4 Projektmanagement Tool . . . 60
3.3 Planung . . . 61
3.3.1 Phasen . . . 61
3.3.2 Meilensteine . . . 62
3.3.3 Zeitplanung . . . 62
3.4 Risiken . . . 64
3.4.1 Technische Risiken . . . 64
3.4.2 Auswertung . . . 65
3.5 Soll-Ist-Zeit-Vergleich . . . 66
3.5.1 Inception . . . 66
3.5.2 Elaboration1 . . . 67
3.5.3 Elaboration2 . . . 67
3.5.4 Construction1 . . . 68
3.5.5 Construction2 . . . 68
3.5.6 Transition . . . 69
3.5.7 Übersicht . . . 69
3.6 Codestatistik . . . 70
3.6.1 Test Coverage . . . 70
3.6.2 Codezeilen . . . 70
3.7 Rational Unified Process (RUP) . . . 71

4 Softwaredokumentation 72
4.1 Installation . . . 73
4.1.1 Redis . . . 73
4.1.2 Keras . . . 73
4.1.3 Docker . . . 74
4.2 Benutzerhandbuch . . . 75
4.2.1 Suche der Fussgängerstreifen . . . 75
4.2.2 Daten visualisieren . . . 77
4.2.3 Challenge erstellen . . . 78
4.2.4 Keras Training . . . 79

Anhang

A Inhalt der CD 80

B Eigenständigkeitserklärung 81

C Glossar **82**

D Literaturverzeichnis **84**

Abbildungsverzeichnis

1 Management Summery Überblick . . . 3
2 Überblick . . . 4
3 Gesamtresultat Kanton Zürich . . . 5

1.1 Haar Feature-based Cascade Classifier . . . 18
1.2 Fast Fourier Transform . . . 19
1.3 Undeutlicher Fussgängerstreifen ohne dominanten Fourierkoeffizienten . . . 19
1.4 Scale-invariant Feature Transform . . . 20
1.5 Deep Learning . . . 20
1.6 Eingezeichnete Strassen in Rapperswil . . . 25
1.7 Beispiel einer Kantendetektion . . . 27
1.8 Queueing . . . 28
1.9 Resultat visualisiert . . . 30
1.10 Resultatmengen . . . 31

2.1 Use Case Diagramm . . . 37
2.2 Sequenzdiagramm Queueing . . . 40
2.3 Sequenzdiagramm Walking . . . 41
2.4 Domain Modell Teil 1 . . . 43
2.5 Domain Modell Teil 2 . . . 44
2.6 Tiles à la Google Maps . . . 49
2.7 QuadTree . . . 49
2.8 Anfertigung Inputbilder . . . 51
2.9 Beispiele für Fussgängerstreifen . . . 52
2.10 Beispiele für Nicht-Fussgängerstreifen . . . 52
2.11 Beispiele NodeMerger . . . 54

3.1 Prof. Stefan Keller . . . 58
3.2 Severin Bühler . . . 58
3.3 Samuel Kurath . . . 59
3.4 Gantt Chart . . . 63
3.5 Inception . . . 66
3.6 Elaboration1 . . . 67
3.7 Elaboration2 . . . 67
3.8 Construction1 . . . 68
3.9 Construction2 . . . 68
3.10 Transition . . . 69
3.11 RUP . . . 71

Tabellenverzeichnis

1.1 Algorithmen Vergleich 21
1.2 Evaluationsmatrix 24
1.3 Resultierende Confusion Matrix Absolute Zahlen 32
1.4 Resultierende Confusion Matrix Prozent Zahlen 32
1.5 Resultate 32

2.1 Aktoren und Stakeholder 37
2.2 Use Case Fully Dressed 39
2.3 MapQuest Application Key 46

3.1 Risiken 64
3.2 Risikoauswertung 65
3.3 Phasen 69
3.4 Test Coverage 70
3.5 Codezeilen 70

Verzeichnis der Entscheide

1.1 Evaluation Suchalgorithmus 21
1.2 Crowdsourcing-System 24
1.3 Queueing Sytem 28
2.4 Main Datei 45
2.5 Eckdaten Anfertigung Inputbilder 51
2.6 NodeMerger Distanzen 54
3.7 PyCharm 60
3.8 GitHub 60
3.9 CircleCI 60
3.10 Jira 60
4.11 Docker 74

0.3 Aufgabenstellung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Gamified Extraction of Crosswalks from Aerial Images

- Studienarbeit im Herbstsemester 2015/2016
- Autoren: Severin Bühler und Samuel Kurath
- Betreuer: Prof. Stefan Keller, Institut für Software, HSR
- Industriepartner: -

Ausgangslage

Der Einsatz von Navigationssystemen beschränkt sich mittlerweile nicht nur auf Autos, sondern wird auch immer mehr von Fussgängern verwendet. Dabei spielen Fussgängerstreifen eine wichtige Rolle. Um eine Route von A nach B optimal für einen Fussgänger zu planen, müssen alle Fussgängerstreifen bekannt sein, um Strassenüberquerungen zu ermöglichen. Ohne die Fussgängerstreifen ist das Routing von Passanten nicht oder nur beschränkt möglich. Dies gilt besonders in Städten und für seh- und gehbehinderte Menschen.

Die Erfassung von solchen Informationen ist ein nicht zu unterschätzendes Problem. Solche Daten werden üblicherweise vor Ort durch lokale Behörden und Fachleute oder von Freiwilligen für das Projekt OpenStreetMap (OSM) erfasst. Nebst dem globalen Navigationssatellitensystem (GPS) gibt es inzwischen weitere Satellitensysteme und Sensoren, die Bilder der Erde liefern (sog. Orthofotos). Diese sind so hochauflösend, dass es möglich geworden ist, Bildobjekte wie Fussgängerstreifen (halb-)automatisch zu erkennen.

Aufgabenstellung

Die Erfassung von Fussgängerstreifen soll mit Hilfe von Orthofotos und Bilderkennungsalgorithmen automatisiert werden. Dabei muss zuerst ein geeigneter Algorithmus evaluiert und in einem zweiten Teil der Arbeit eine Software zur (halb-)automatischen Datenverarbeitung geschrieben werden. Als Inputdaten dienen einerseits Orthofotos und andererseits Strassenachsen aus OpenStreetMap. Als Output werden Koordinaten erwartet allenfalls mit Zusatzinformationen (Genauigkeit). Diese Daten müssen validiert werden. Dies geschieht durch die Verwendung von einem - ebenfalls zu evaluierenden - Crowdsourcing-System, bei dem Freiwillige die gefundenen Daten in OpenStreetMap einfügen (beispielsweise MapRoulette, To-Fix oder Kort.ch).

Ziele

- Evaluation eines effizienten Algorithmus zur Erkennung von Fussgängerstreifen auf Orthofotos.
- Automatische Verarbeitung von Orthofotos.
- Extraktion der Koordinaten von Fussgängerstreifen aus Orthofotos (Kanton Zürich, optional Europa oder mehr).
- Evaluation des Crowdsourcing-System zur Daten-Validierung und Übertragung in OSM.
- Erstellung einer Challenge für das Crowdsourcing-System anhand der gesammelten Daten.

Lieferobjekte

1. Dokumentation, inkl. technischer Bericht und Software Engineering-Projekt (deutsch).
2. Fussgängerstreifen- Daten als Resultat des Erkennungssoftware innerhalb des Kanton Zürich (optional: europa- oder weltweit).
3. Challenge auf evaluiertem Crowdsourcing-System bereitgestellt und eingereicht.
4. Die vom Studiengang geforderten Lieferobjekte: Dokumentation, Management Summary, Abstract, Poster (digital).
5. Software (englisch) einfach installierbar (z.B. Docker) mit Installationsanleitung.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Vorgaben/Rahmenbedingungen

- Serverseitig: Python (zweite Priorität Java)
- Clientseitig bzw. Verwaltungs- und Testsoftware: ggf. Python
- Daten: Daten der öffentlichen Hand (z.B. Kanton Zürich) sind vorhanden
- Wöchentliche Meetings mit vorbereiteten Unterlagen (Ausnahmen werden zusammen vereinbart)

Dokumentation

Die Arbeit umfasst folgende Inhalte:

- Abstract (deutsch und englisch), Management Summary, Aufgabenstellung
- Technischer Bericht
- Software Engineering-Projektdokumentation
- Anhänge (Literaturverzeichnis, CD-Inhalt)

Weitere Angaben:

- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind (einheitliche Nummerierung).
- Die Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen (nicht ausschliesslich Wikipedia-Links auflisten).
- Dokumentation des Projektverlaufes, Planung etc.
- Weitere Dokumente (z.B. Kurzbeschreibung, Eigenständigkeitserklärung, Nutzungsrechte) gemäss Vorgaben des Studiengangs und Absprache mit dem Betreuer.

Form der Dokumentation:

- Bericht (Struktur gemäss Beschreibung) gebunden (2 Exemplare), inkl. je einer beschrifteten CD plus 1 Exemplar für die Abteilung/Archivierung.
- Alle Dokumente und Quellen der erstellten Software auf CDs.

Bewertung

Es gelten die üblichen Regelungen zum Ablauf und zur Bewertung der Studienarbeit des Studiengangs Informatik mit besonderem Gewicht auf moderne Softwareentwicklung wie folgt:

- Projektorganisation (Gewichtung ca. 1/5)
- Bericht, Gliederung, Sprache (Gewichtung ca. 1/5)
- Inhalt inkl. Code (Gewichtung ca. 2/5)
- Gesamteindruck inkl. Kommunikation mit Industriepartner (Gewichtung ca. 1/5). Ein wichtiger Bestandteil der Arbeit ist, dass eine lauffähige, getestete Software abgeliefert wird (inkl. getesteter Installationsanleitung).

Weitere Beteiligte

Keine.

Kapitel 1

Technischer Bericht

1.1 Einführung

1.1.1 Vision

Aktuell Die Navigation von Fussgänger wird immer beliebter, dabei spielen Fussgängerstreifen ein wichtige Rolle. Leider sind diese nur spärlich erfasst. Auch die Behörden selbst haben keine kompletten Daten zu den Standorte. So hat zum Beispiel die Stadt Zürich das Projekt Zebra-Safari lanciert, bei dem bis Ende 2016 alle Fussgängerstreifen erfasst und beurteilt werden sollen. Dieser Vorgang ist jedoch sehr arbeitsintensiv und wird von Hand durchgeführt.

Vision Es soll eine Applikation entstehen, mit der die aufwändige Problematik der Erfassung der Fussgängerstreifen automatisiert werden kann. Dies soll mit Hilfe von Bilderkennungsalgorithmen auf Orthofotos (Satellitenbildern) ermöglicht werden.

1.2 Stand der Technik

Um abzuklären, ob es schon Arbeiten gab, die ähnliche Probleme lösen, nahmen wir uns im Rahmen der Semesterarbeit Zeit für ein Literaturrecherche. Dabei gingen wir auf die HSR Bibliothek und deren Mitarbeiter zu.

1.2.1 Literaturrecherche

Suchquellen

Folgende Quellen wurden uns empfohlen, um Recherchen in diesem Umfeld durchzuführen:

- NEBIS [DRB15]
- IEEXplore [IEE15]
- Google Scholar [Goo15]

Auswertung

Bei der Recherche stiessen wir auf verschiedene Projekte, die sich mit der Problematik des Erkennens von Fussgängerstreifen auseinander setzen. Leider sind diese Arbeiten eher im Bereich der Bilderkennung für die Steuerung von autonom fahrenden Autos/Robotern angesiedelt. Arbeiten die treffender sind, werden im Anschluss angeführt.

Extraction of Road Markings from Aerial Images

Yuichi Ishino und Hitoshi Saji (Japan, 2008)[IS08]

An der Univerisität Shizuoka in Japan gab es vor einigen Jahren eine Arbeit zur Erkennung von Fussgängerstreifen und Mittellinien (Traffic Lane Lines) auf Orthofotos.

Ihr Algorithmus befolgt dabei folgende Strategie: Der Algorithmus geht den Strassen entlang und richtet die Bilder so aus, dass die Fussgängerstreifen immer vertikal zur Achse laufen. Danach wird eine sogenannte Binarization durchgeführt. Es setzt alle Pixel unter einem Schwellwert auf 0 (weiss) und alle Pixel darüber auf 1 (schwarz). Es wurden zwei Schwellwerte zuvor berechnet, einmal für sonnige und einmal für schattige Bilder. Mit den Annahmen, dass die Strasse schwarz/grau und der Fussgängerstreifen leuchtend weiss sind, sieht man nun ein gleichmässiges Muster in der Helligkeitsverteilung des Bildes. Ein Fouriertransformation würde eine saubere Frequenz liefern.

Die Arbeit von Ishino und Saji geht von einigen Grundannahmen und Vorraussetzungen aus, die die Erkennung sehr erleichtern:

- Die Fussgängerstreifen sind immer gerade und werden durch keine Inseln unterbrochen.
- Die Auflösung der Bilder ist genug gross, um das Streifenmuster ohne Probleme zu erkennen.
- Der Fussgängerstreifen ist immer deutlich heller als die Strasse selbst.
- Der Streifen werden durch keine Hindernisse wie Bäume, Autos verdeckt oder beeinflusst.
- Die Bilder sind zuvor in die Kategorien schattig und sonnig eingeteilt worden. Auf ihnen wird mit verschiedenen Schwellwerten gearbeitet.
- Die Strassen müssen die Fussgängerstreifen immer vertikal schneiden.

Schlussfolgerung Die Arbeit der Universität von Shizuoka verfolgte einen ähnlichen Ansatz, den wir mit der Fouriertransformation in Betracht ziehen. Leider gehen die Dokumentverfasser von einigen Grundannahmen aus, die sich nicht mit unserer Arbeit decken. Man kann fast schon von Laborbedingungen sprechen. Doch gibt es einigen Techniken, die sich auch für unsere Arbeit verwenden lassen. Diese sind im Fazit aufgeführt.

Segmentation of Occluded Sidewalks in Satellite Images

Turgay Senlet und Ahmed Elgammal, The State University of New Jersey, USA (2012)[SE12]

Das Projekt von Turgay Selent und Ahmed Elgammal setzte sich mit der Erkennung von primär Gehwegen (sidewalks) und Fussgängerstreifen auf Satellitenbildern auseinander.

Dabei waren die Hauptprobleme, dass viele Gehwege von Bäumen oder Schatten verdeckt werden. Um diesem Problem Herr zu werden, benutzten sie einen Farbklassifizierer. Um Fussgängerstreifen zu klassifizieren, stellten sie eine Sammlung an Frequenzen in allen möglichen Winkeln zusammen.

Leider wird im Artikel zu dieser Arbeit nicht weiter auf die Erkennungsmethoden eingegangen.

Fazit

Aus allen Arbeiten konnten wir doch einige Techniken finden, die uns die Erkennung erleichtern könnten. Diese sind hier aufgelistet:

- Binarization image
- Median Filter (für Verbesserung der Bildqualität von ungenauen Bildern)

1.3 Evaluation Suchalgorithmus

Die Evaluation verschiedener Algorithmen zur Erkennung von Fussgängerstreifen stellte ein wichtiger Teil unserer Arbeit dar. Nach Absprachen mit unseren Dozenten und Recherchen im Internet sind wir auf die unteren Methoden zur Bilderkennung gestossen. Um die Kandidaten zu vergleichen, griffen wir auf das Werkzeug der Confusion Matrix (Wahrheitsmatrix) zurück.

1.3.1 Algorithmen Vergleich

Um einen neutralen Vergleich der verschiedenen Erkennungsmethoden zu erreichen, haben wir alle Methoden testweise implementiert und mit ihnen eine Erkennung im Raum Rapperswil durchgeführt.

Wir haben mit folgenden Eckdaten gearbeitet:

Bounding Box (Rapperswil): (8.814650, 47.222553, 8.825035, 47.228935)
Anzahl Fussgängerstreifen: 37

Haar Feature-based Cascade Classifier

		Vorhergesagt	
		Position **ist** Fussgängerstreifen	Position ist **kein** Fussgängerstreifen
Tatsächlich	Position **ist** Fussgängerstreifen	3 (TP)	34 (FN)
	Position ist **kein** Fussgängerstreifen	53 (FP)	2562 (TN)

Abbildung 1.1: Haar Feature-based Cascade Classifier

Auf der Suche nach möglichen Algorithmen, welche Objekte auf Bildern erkennen, stiessen wir auf den Haar Feature-based Cascade Classifier. Dieser erlaubt es Objekte unabhängig von der Drehung oder Skalierung wieder zu erkennen. In einer Erweiterung von OpenCV ist der Algorithmus implementiert und kann wie in dem Beispiel [Bal15] von Thorsten Ball verwendet werden.

Für das Trainieren braucht es sehr viele Bilder und noch mehr Rechenleistung. Um diese Hürden zu überwinden, schrieben wir ein Python Skript, welches uns Bilder von Fussgängerstreifen lieferte. Dabei konnten wir auf OpenStreetMap Daten zurückgreifen, welche die Koordinaten von Fussgängerstreifen lieferten. Weiter stellte die HSR ihren Dev-Server zur Verfügung, um die enorme Rechenleistung abzudecken. Leider wurde auch mit diversen unterschiedlichen Trainingsdaten kein brauchbares Resultat erzielt. Eine Schwierigkeit beim trainieren war, dass man kein Feedback bekam, was gutes oder eher schlechtes Bildmaterial für das Training ist.

Fast Fourier Transform

		Vorhergesagt	
Tatsächlich		Position **ist** Fussgängerstreifen	Position ist **kein** Fussgängerstreifen
	Position **ist** Fussgängerstreifen	28 (TP)	8 (FN)
	Position ist **kein** Fussgängerstreifen	7 (FP)	2562 (TN)

Abbildung 1.2: Fast Fourier Transform

Die Orthofotos entlang der Strasse wurde so ausgerichtet, dass die Strasse horizontal zum Bild liegt. Der Fussgängerstreifen hat nun die Eigenschaft, dass wenn eine Fourier Transformation vertikal zum Bild angewendet wird, ein eindeutiges Frequenzmuster entstehen sollte. Genau genommen ist der 8. Fourierkoeffizient gegenüber seinen Nachbar erhöht (Je nach Bildqualität auch der 7. oder 9.). Die Methode versagt jedoch, sobald der Fussgängerstreifen undeutlich wird, Objekte wie Autos auf dem Fussgängerstreifen sind oder andere Gegenstände diese dominante Frequenz auch beinhalten.

Abbildung 1.3: Undeutlicher Fussgängerstreifen ohne dominanten Fourierkoeffizienten

Scale-invariant Feature Transform

		Vorhergesagt	
		Position **ist** Fussgängerstreifen	Position ist **kein** Fussgängerstreifen
	Position **ist** Fussgängerstreifen	3 (TP)	34 (FN)
	Position ist **kein** Fussgängerstreifen	195 (FP)	2562 (TN)

Abbildung 1.4: Scale-invariant Feature Transform

Scale-invariant Feature Transform (SIFT) ist ein Algorithmus, welcher Merkmale von Bildern beschreibt. Seine Stärken sind, dass es keine Probleme mit Translation, Rotation oder Skalierung gibt. Die Idee dies Verfahrens ist jedoch ein Objekt auf einem Bilder wieder zu erkennen. Mit Input, welches eine unbekannte Situation zeigt, kommt der Algorithmus nicht klar.
OpenCV bietet eine Implementation von SIFT, welche wir nutzten um die Evaluation durchzuführen. Das Resultat war für unsere Problemstellung alles andere als ausreichend.

Deep Learning - Convnet

		Vorhergesagt	
Tatsächlich		Position **ist** Fussgängerstreifen	Position ist **kein** Fussgängerstreifen
	Position **ist** Fussgängerstreifen	35 (TP)	2 (FN)
	Position ist **kein** Fussgängerstreifen	0 (FP)	2562 (TN)

Abbildung 1.5: Deep Learning - Convnet

Das Convolutional Neural Network hat die Fussgängerstreifen mit Abstand am besten erkannt. Anzumerken ist hier, dass die False Positive (FP) Rate von 0 ein schlichtes Zufallsergebnis ist. Wir haben diese Technik nach der Evaluation auf grösseren Flächen innerhalb der Stadt Zürich getestet und sind auf 1 FP pro 1000 Bildern gekommen. Eine Beschreibung der Technik des Convnets können sie dem Abschnitt 1.5 auf der Seite 25 entnehmen.

1.3.2 Auswertung

Damit die Auswertung verständlich ist, wird hier noch auf die Berechnung und die angeführte Legende verwiesen.

Legende

TP:	Zahl der richtig positiven Klassifikationen
FP:	Zahl der falsch positiven Klassifikationen
TN:	Zahl der richtig negativen Klassifikationen
FN:	Zahl der falsch negativen Klassifikationen

Berechnung

Trefferquote	=	TP / (TP+FN)
Richtigkeit	=	(TP + TN) / (TP + FP + TN + FN)
Relevanz	=	TP / (TP + FP)

Algorithmus	Tefferquote	Richtigkeit	Relevanz
Haar Feature-based Cascade Classifier	0.08	0.97	0.05
Scale-invariant feature transform	0.08	0.91	0.02
Fast Fourier Transform	0.77	0.99	0.8
Deep learning	0.95	0.99	1.0

Tabelle 1.1: Algorithmen Vergleich

Entscheid 1. Evaluation Suchalgorithmus

An dieser Stelle ist zu erwähnen, dass Bilderkennung im Allgemeinen ein nicht triviales Problem ist. Man hat mit den unterschiedlichsten Schwierigkeiten zu kämpfen, wie der Qualität oder die Belichtung der Bilder. Das führte dazu, dass nur mit dem Fast Fourier Transform und dem Deep Learning Ansatz Resultate erziehlt wurden, welche ein brauchbares Ergebnis lieferten. Deep Learning ist jedoch der klare Favorit und sticht insbesondere beim False Positive Wert hervor. Deshalb entschieden wir uns unseren Fokus auf diesen Algorithmus zu legen.

1.4 Evaluation Crowdsourcing-System

Da Daten nicht automatisch in OpenStreetMap integriert werden dürfen, wird ein Crowdsourcing-System benötigt. In diesem überprüfen Benutzer auf spielerische Art und Weise, ob die Daten korrekt sind.
Es gibt verschiedene Produkte in diesem Bereich, somit haben wir als Teil unserer Arbeit eine Evaluation zweier relevanter Kandidaten durchgeführt.

Kandidaten

- MapRoulette [vE15a]
- To-Fix [Lab15]

1.4.1 MapRoulette

MapRoulette verwendet für ihre Challenges und Tasks ein einfaches JSON Format. Erstellt werden Challenges mittels POST und mit PUT können diese upgedatet werden.

Beispiel Challenge

Erstellen: POST /api/admin/challenge/<slug>
Updaten: PUT /api/admin/challenge/<slug>
Challenge JSON:

```
{
  "title": "Repair Motorways",
  "description": "Repair all motorways",
  "blurb": "The idea is to repair all motorways",
  "help": "Repair the ways where it is broken on the map",
  "instruction": "Look at the map for broken pieces.",
  "active": true,
  "difficulty": 2
}
```

Beispiel Task

Erstellen:	POST /api/admin/challenge/<slug>/task/<task_identifier>
Updaten:	PUT /api/admin/challenge/<slug>/task/<task_identifier>

Challenge JSON:

```
{
  "instruction" : "This is a hard task!",
  "geometries" : {
    "type": "FeatureCollection",
    "features": [
      { "type": "Feature",
        "geometry":
        { "type": "Point",
          "coordinates":[-41.4710170873565,31.235521774136]
        },
        "properties": {"osmid": 12345}
      }
    ]
  }
}
```

1.4.2 To-Fix

To-Fix verwendet für ihre Tasks ein CSV Format, welches direkt über das grafische Benutzerinterface publiziert werden kann.

Beispiel CSV

```
object_type,object_id,st_astext
way,51446110,POINT(-94.4176451 43.3273692)
way,187403368,POINT(32.9369086 2.1997495)
way,220866128,POINT(-68.5 49.647521)
way,223982938,POINT(18.4823301 59.6732909)
way,109819283,POINT(-83.1888421 40.0485764)
```

1.4.3 Auswertung

Um die beiden Kandidaten zu vergleichen, haben wir eine Evaluationsmatrix erstellt. Dabei haben wir diverse für uns relevante Kriterien erarbeitet und diese jeweils auf einer Skala von 1 bis 10 gewichtet. In einem zweiten Schritt haben wir den Kandidaten für die jeweiligen Kriterien Punkte vergeben.

Kriterium	Gewicht	Maproulette	Resultat	To-Fix	Resultat
Challenge ist leicht erstellbar	5	6	30	7	35
Challenge ist leicht publizierbar	7	8	56	8	56
Anbieter ist relevant bei der Community	8	8	64	4	32
Dokumentation	7	5	35	5	35
Kontaktperson	5	5	25	6	30
Total	32	32	210	30	188

Tabelle 1.2: Evaluationsmatrix

Entscheid 2. Crowdsourcing-System

Beide Kandidaten haben Vor- und Nachteile, wie aus der Evaluationsmatrix ersichtlich ist. Für uns ist das wichtigste Kriterium, wie relevant der Anbieter bei der Community ist, was sich dann auch im Resultat stark ausgewirkt hat. Da MapRoulette Challenges gerne abgearbeitet werden, haben wir uns für diesen Kandidaten entschieden

1.5 Umsetzungskonzept

Die Grundidee für die Umsetzung der Fussgängerstreifen Erkennung lässt sich in folgenden Punkten zusammenfassen:

1. Download der Strassen- und Fussgängerstreifenkoordinaten
2. Download von Orthofotos auf OpenStreetMap Zoomlevel 19
3. Anfertigen von einheitlichen Bildern entlang der Strassen
4. Fussgängerstreifenerkennung mit Hilfe des Convnets
5. Vergleich bestehende Fussgängerstreifen mit neu gefundenen
6. Parallelisierung des Erkennungsprozesses
7. Daten in eine Maproulette Challenge umwandeln

1.5.1 OpenStreetMap

Um Daten von OpenStreetMap zu erhalten, kann man eine OpenStreetMap Web API ansprechen. Mapquest [Inc15a] stellt eine solche OpenStreetMap API zur Verfügung. Mit dieser Schnittstelle lassen sich via HTTP GET Abfragen starten. Eine Bounding Box und die entsprechenden OpenStreetMap Tags dienen als Input. Man erhält danach alle Daten in einem einfach zu interpretierenden XML Format. Mit der gleichen Abfrage lassen sich sogar Fussgängerstreifen und Strassen zur selben Zeit herunterladen.

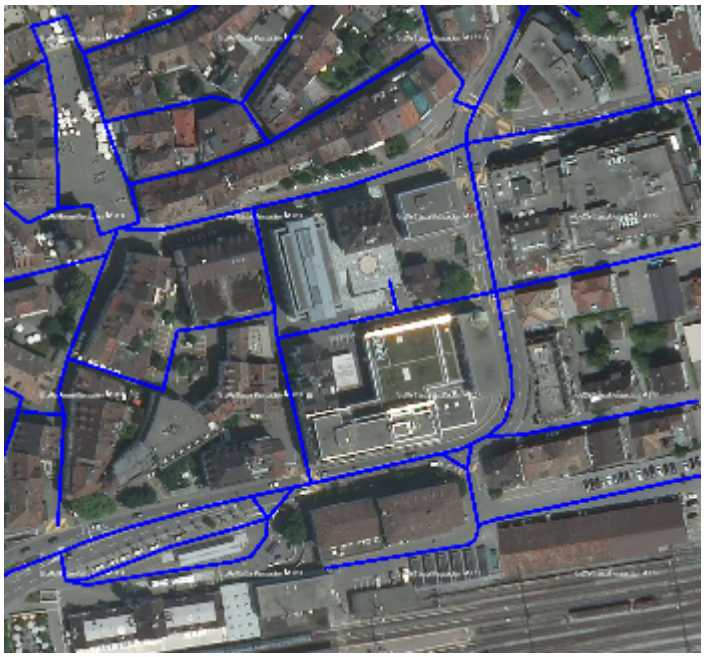

Abbildung 1.6: Eingezeichnete Strassen in Rapperswil

1.5.2 Bing

Um an die Orthofotos zu kommen, gab es zu Beginn des Projektes mehrere Lösungen:

1. Offizielle Microsoft Bing REST Service
2. Direkter Download über Bing Maps
3. Benutzung der Orthofotos von der HSR

Als aller erstes haben wir versucht, die Orthosfotos über den offizielle Microsoft Bing REST Service [Mic15a] zu download. Jedoch beschränkt Microsoft den API Zugriff auf 50'000 Transaktionen pro Tag und 125'000 Transaktionen pro Jahr, was uns bei geschätzten 7 Millionen Tiles in der Schweiz nicht reichen würde.

Nachdem diesem ersten Versuch sind wir auf den direkten Download der Othosfotos umgestiegen. Wenn man mit dem Internet Browser auf Bing Maps zugreift, werden die Tiles mithilfe von Javascript von den Bing Servern geladen. Microsoft benutzt das Quadtree Format, um die entsprechenden Tiles anzusprechen. Mithilfe des Projekts Tiles à la Google Maps [Př15] und der dort zur Verfügung gestellten Python Library, können wir Bounding Boxen in Quadtrees umwandeln.

Die Orthofotos, welche im Besitz der HSR sind und die nur die Schweiz umfassen, wären die letzte Lösung gewesen, wenn die anderen Möglichkeiten versagt hätten.

1.5.3 Convolutional Neural Network

Die Evaluation des Suchalgorithmus hat einen klaren Sieger ergeben. Das Convolutional[1] Neural Network (hier weiter als Convnet bezeichnet) liefert mit Abstand die besten Resultate innerhalb unserer Test Bounding Box in Rapperswil.

Geschichte

Bis noch vor einigen Jahren wurden neuronale Netze zur Bilderkennung grösstenteils ignoriert. Das Convnet selbst wurde schon 1980 von einem Japaner namens Fukushima erfunden, jedoch erhielt es keine grössere Beachtung. Hauptgrund dafür war der Rechenhunger für das Training des Netzes. Gedreht hat sich das Ganze erst als 2012 genügend Rechenleistung mithilfe von Grafikkarten zur Verfügung stand. Ab diesem Zeitpunkt erzielte diese wieder gefundene Technik Bestleistungen in vielen Bereichen der Bilderkennung. Bis heute ist das Convnet die präziseste Technik zur Erkennung von Gegenständen in Fotos.

Funktionsweise

Ein künstliches neuronales Netz besteht aus einer grossen Anzahl von simulierten Neuronen. Mit verschiedenen Techniken aus der Statistik und Mathematik kann so ein Input auf einen Output gemappt werden. In unserem Fall wollen wir ein Bild auf eine der Kategorien Crosswalk oder Non-Crosswalk mappen. Dies wird in der Fachliteratur auch Klassifikation von Bildern genannt.

Wie der Name schon sagt, besteht das Convolutional Neuronale Netz aus vielen verschiedenen Faltungsfilter. Ein Faltungsfilter transformiert ein Bild so, dass ein spezifisches Muster auf dem Bild markiert wird. Ein gutes Beispiel für einen Filter ist die Technik der Kantendetektion. Die Kantendetektion markiert nur die Kanten auf dem Bild und ignoriert den restlichen Inhalt.

[1] Convolutional ist Englisch und heisst auf Deutsch Faltung

Abbildung 1.7: Beispiel einer Kantendetektion

Ein Convolutional Neuronales Netz lernt nun selbst, welchen Faltungsfilter er anwendet, um das Problem möglichst gut zu lösen. Die extrahierten Muster dienen als Features und werden nun von einem einfachen Neuronalen Netz klassifiziert. Die Ausgabe besteht aus aus zwei Werten zwischen 0 und 1. Sie stellen die Wahrscheinlichkeiten der entsprechenden Klasse dar.

Keras

Das Projekt Keras [Cho15] stellt eine einfache Library zum Entwerfen und Trainieren von Neuronalen Netzen zur Verfügung. Es bietet modulare Funktionen für Convolutional Neuronale Netzwerke und Rekurrente Netzwerke an. Mithilfe eines Flags kann das Training einfach auf die Grafikkarte ausgelagert werden, was bei solchen Algorithmen unerlässlich ist.

1.5.4 Parallelisierung

Zu Beginn unserer Arbeit unterschätzten wir die enorme Datenmenge in Form von Orthofotos (Beispiel Schweiz: 7.1 Millionen Bilder à $5820m^2$ Fläche pro Bild). Weiter wird auch sehr viel Rechenleistung für die Erkennung der Fussgängerstreifen auf den Bilder benötigt. Um diesen nicht trivialen Problemen Herr zu werden, setzten wir auf eine Parallelisierungsstratege mit Hilfe einer Queue.

Entscheid 3. Queueing Sytem

Den Entscheid für Redis [San15] in Kombination mit RQ [Dri15] haben wir während eines Meeting mit Hilfe von Mitarbeitern des Institut für Software erarbeitet. RQ ist eine relativ einfach zu verwendende Library, welche Redis (Key Value Store) als Queue einsetzt. Durch die Einfachheit und die gute Integration in Python haben wir uns für diesen Lösungsweg entschieden.

Aufbau

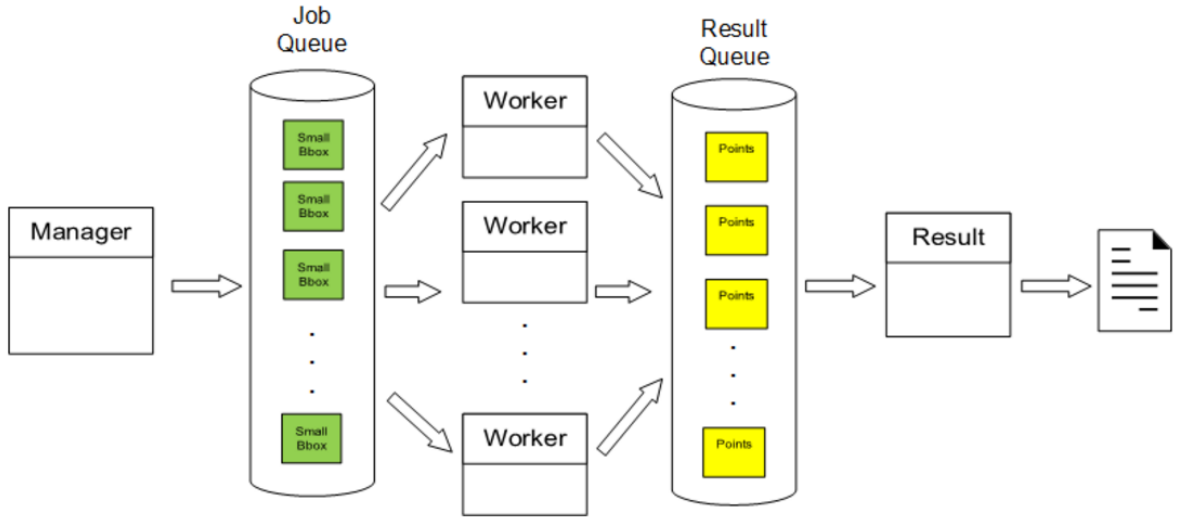

Abbildung 1.8: Queueing

Auf der Abbildung ist zu sehen, dass wir für das Verarbeiten der Jobs auf zwei Queues setzten, eine die die abzuarbeitenden Bounding Boxes beinhaltet und eine weitere für das Sammeln der Resultate. Der genau Ablauf gestaltet sich wie folgt:

1. Manager wird aufgerufen mit der grosse Input Bounding Box.
2. Manager teilt Bounding Box auf.
3. Kleine Bounding Boxes werden als Jobs in die Job Queue geladen.
4. Jobs werden von den Worker aus der Queue geholt.
5. Worker arbeiten kleine Bounding Boxen ab.
6. Worker stellt die gefunden Punkt in Result Queue.
7. Result Worker holt die gefunden Punkt aus der Result Queue und speichert diese in einer JSON Datei ab.

1.5.5 MapRoulette

Maproulette ist ein Crowdsourcing-System, welches genutzt wird um mit Hilfe von Challenges Daten in OpenStreetMap einfliessen zu lassen, diese kontrolliert oder auch erweitert. Den Entscheid für diesen Anbieter haben wir unter dem Abschnitt 1.4.1 auf der Seite 22 getroffen.

Daten in Tasks umwandeln

Am Ende des Suchprozesses werden alle gefundenen Fussgängerstreifen in einer JSON Datei abgespeichert. MapRoulette benötigt für die Tasks jedoch ein GeoJSON Format, somit mussten wir diese Daten noch passend umwandeln.

Challenge Der Text für die Challenge ist in englisch zu verfassen und sollte folgende Punkte beinhalten:

- Title
- Blurb (Einzeiler, eine Art Untertitel)
- Description
- Help
- Instruction

Task Unsere Task beinhalte hauptsächlich nur die Position des zu verifizierenden Fussgängerstreifens, sowie eine kurze Beschreibung. Der Aufbau gliedert sich folgendermassen:

- Identifier (Eindeutiger 72 Character String)
- Geometries (Position des Fussgängerstreifens)
- Instruction (Beschriebt den Task)

1.6 Resultate

1.6.1 Zielerreichung

Das Projekt Extraction of Crosswalks from Aerial Images barg einige Hürden, von der Schwierigkeit der Bilderkennung, dem Anbinden diverser Schnittstellen und die parallele Verarbeitung grosser Datenmengen.
Als Resultat der Arbeit entstand eine Applikation, welche auf Orthofotos Fussgängerstreifen erkennt und deren Koordinaten speichert. Im Bild unterhalb sind die vom Algorithmus gefundenen Fussgängerstreifen der Ostschweiz visualisiert.

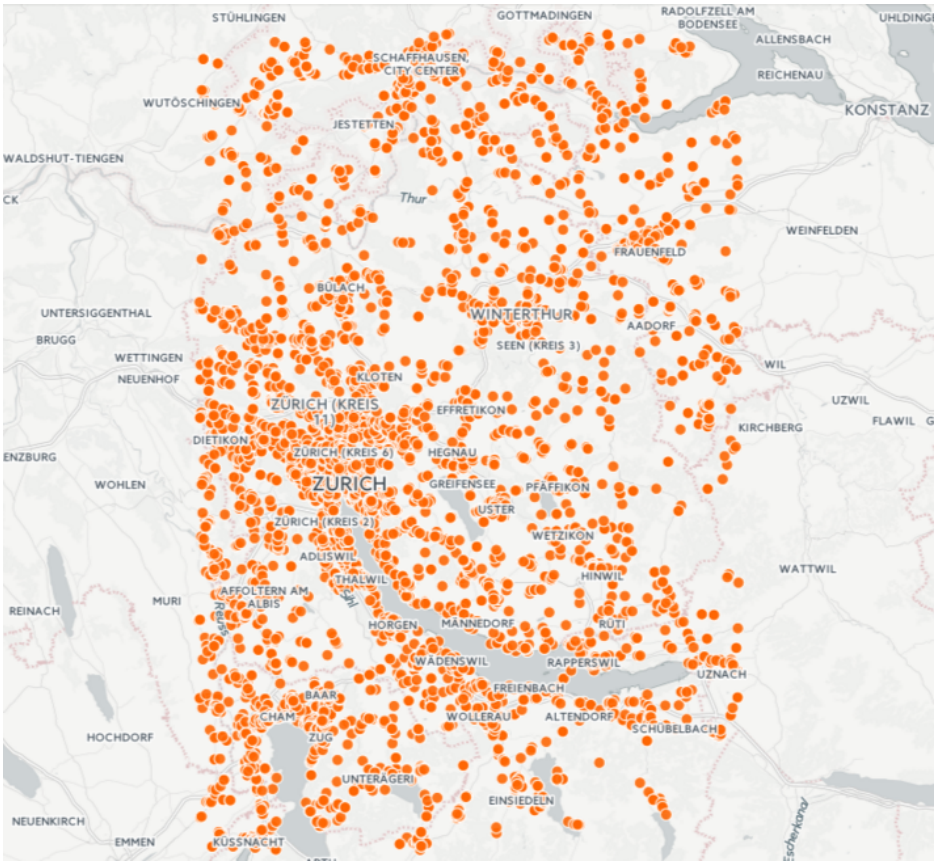

Abbildung 1.9: Resultat visualisiert, Zürich, Zug und Teile von St. Gallen

1.6.2 Resultierende Dateien

Aus dem Projekt sind mehrere JSON Datei hervorgegangen, welche auf der CD im Resultat/Daten Ordner zur Verfügung stehen. Diese Datei beinhalten die vom Projekt gefundenen Fussgängerstreifen abzüglich der bestehenden Fussgängerstreifen in OpenStreetMap.

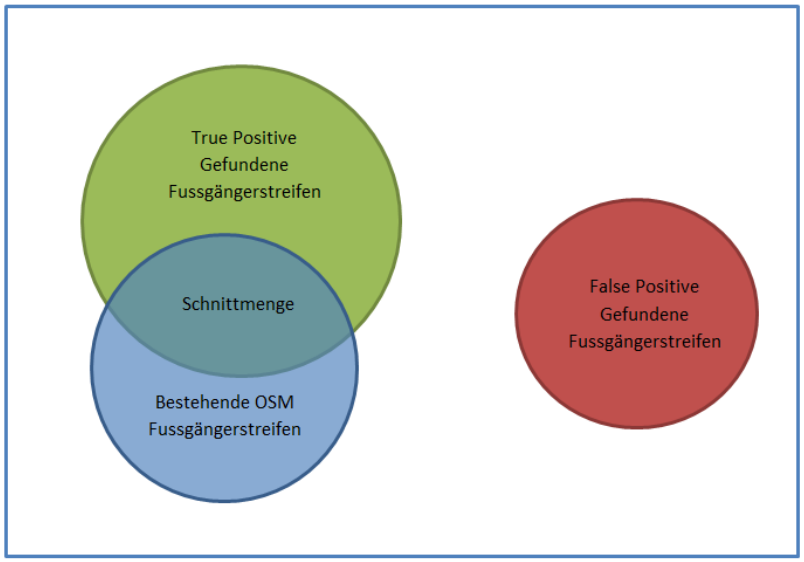

Abbildung 1.10: Resultatmengen - Die vom Crosswalk Detector gefundenen Resultate sind Grün und Rot dargestellt. Die OpenStreetMap Daten sind Blau. Die Projektresultate beinhalten die Koordinaten Rot + Grün - Blau.

Wahrheitstabelle mit Kennzahlen

Um saubere Kennzahlen zur Erkennungrate zu erhalten, haben wir den Algortihmus auf die Fläche zwischen Zollikon und Zürich Zoo angewendet und die Daten in einer Confusion Matrix zusammengefasst.

	Positive (Crosswalk Detector)	Negative (Crosswalk Detector)
True (Human prediction)	187	16364
False (Human prediction)	19	39

Tabelle 1.3: Resultierende Confusion Matrix Absolute Zahlen - BBox (47.355633, 8.543026), (47.372811, 8.570957)

	Positive (Crosswalk Detector)	Negative (Crosswalk Detector)
True (Human prediction)	83%	-
False (Human prediction)	9%	17%

Tabelle 1.4: Resultierende Confusion Matrix Prozent Zahlen - BBox (47.355633, 8.543026), (47.372811, 8.570957)

In Worten ausgedrückt, sind die Zahlen folgendermassen zu interpretieren:

- 83% aller visuell sichtbaren Fussgängerstreifen wurden erkannt.
- 17% der Fussgängerstreifen werden nicht erkannt.
- 9% der erkannten Fussgängerstreifen sind falsch.

Ziel der Aufgabenstellung

Ziel	Resultat
Evaluation eines effizienten Algorithmus zur Erkennung von Fussgängerstreifen auf Orthofotos.	Diverse Algorithmen wurden evaluiert, mit dem Deep Learning Ansatz wurde ein klarer Favorit ermittelt. Mehr dazu unter dem Abschnitt 1.3 auf der Seite 18.
Automatische Verarbeitung von Orthofotos.	Es wird automatisch auf Bilder von Bing Maps zugegriffen.
Extraktion der Koordinaten von Fussgängerstreifen aus Orthofotos (Kanton Zürich, optional Europa oder mehr).	Zürich konnte von der Applikation verarbeitet werden, weiter wurde die Suche auf die Ostschweiz ausgebaut.
Evaluation des Crowdsourcing-System zur Daten-Validierung und Übertragung in OSM.	Bei der Evaluation des Crowdsourcing-Systems setzte sich MapRoulette durch die Bekanntschaft bei der Community durch. Mehr dazu unter dem Abschnitt 1.4 auf der Seite 22.
Erstellung einer Challenge für das Crowdsourcing-System anhand der gesammelten Daten.	Eine Challenge wurde für MapRoulette generiert und publiziert.

Tabelle 1.5: Resultate

Die in der Aufgabenstellung formulierten Ziele konnten alle in einem angemessenem Rahmen erreicht werden.

1.6.3 Persönlicher Bericht

Mit dem Resultat des Projektes sind wir äusserst zufrieden. Es wurde viel neues dazugelernt, sowohl in technischen Bereichen, wie auch in der Teamkommunikation und dem Projektmanagement.

Neu erlernte Technologien

- Python
- Docker
- Anwendung Neuronale Netze (Keras)
- OpenCV
- Latex

Daten

Weiter hatten wir es mit einer grossen Datenmenge zu tun. In Zahlen gesprochen:

Durchsuchte Fläche:	$8800 km^2$
Anzahl Tiles:	1'512'028 (à $5820 m^2$ pro Tile)
Anzahl Trainingsbilder	36'400 RGB Bilder 50 x 50 Pixel

Einerseits stellte die Datenbeschaffung schon ein grosse Herausforderung dar, andererseits war auch die parallele Verarbeitung nicht trivial. Mit der Queueing Lösung konnten wir dem jedoch erfolgreich entgegenwirken.

Das Arbeiten an diesem Projekt mit all diesen neuen Technologien hat uns viel Spass gemacht. Die Motivation war dadurch auch permanent hoch.

1.6.4 Dank

Für die Betreuung während des ganzen Projektes möchten wir uns besonders bei Herr Prof. Stefan Keller Bedanken, der immer ein offenes Ohr für Problem aller Art hatte und uns freien Spielraum für die Suche nach einer optimalen Lösung gewährte.
Weiter geht auch ein Dank an Herr Prof. Dr. Guido Schuster, welcher uns in Bezug auf den Erkennungsalgorithmus unterstützte.
Nicht zu vergessen sind die Mitarbeiter des Geometalabs (Teil des Institut für Software der HSR), welche uns bei der Findung einer passenden Lösung für die parallele Verarbeitung der Daten halfen.
Und zu guter Letzt möchten wir auch noch Manuel Roth und Lukas Martinelli Danken, die uns in Fragen rund um Tiles und deren Umrechnung unterstützten.

Kapitel 2

Software Projektdokumentation

2.1 Vision

Die Vision ist unter dem Abschnitt 1.1.1 auf der Seite 15 zu finden.

2.2 Anforderungsspezifikation

2.2.1 Use Case

Aktoren und Stakeholder

Aktor	Tätigkeit
User	Startet CrosswalkDetector mit Bounding Box als Eingabeparameter
CrosswalkDetector	Erkennt Fussgängerstreifen
TileProvider	Stellt Orthofotos für die Erkennung der Fussgängerstreifen zur Verfügung.
OSMProvider	Stellt Strassen - und Fussgängersteifen Informationen zur Verfügung.
JSON File	Speicherort für die Positionen der ermittelten Fussgängersteifen.

Tabelle 2.1: Aktoren und Stakeholder

Use Case Diagramm

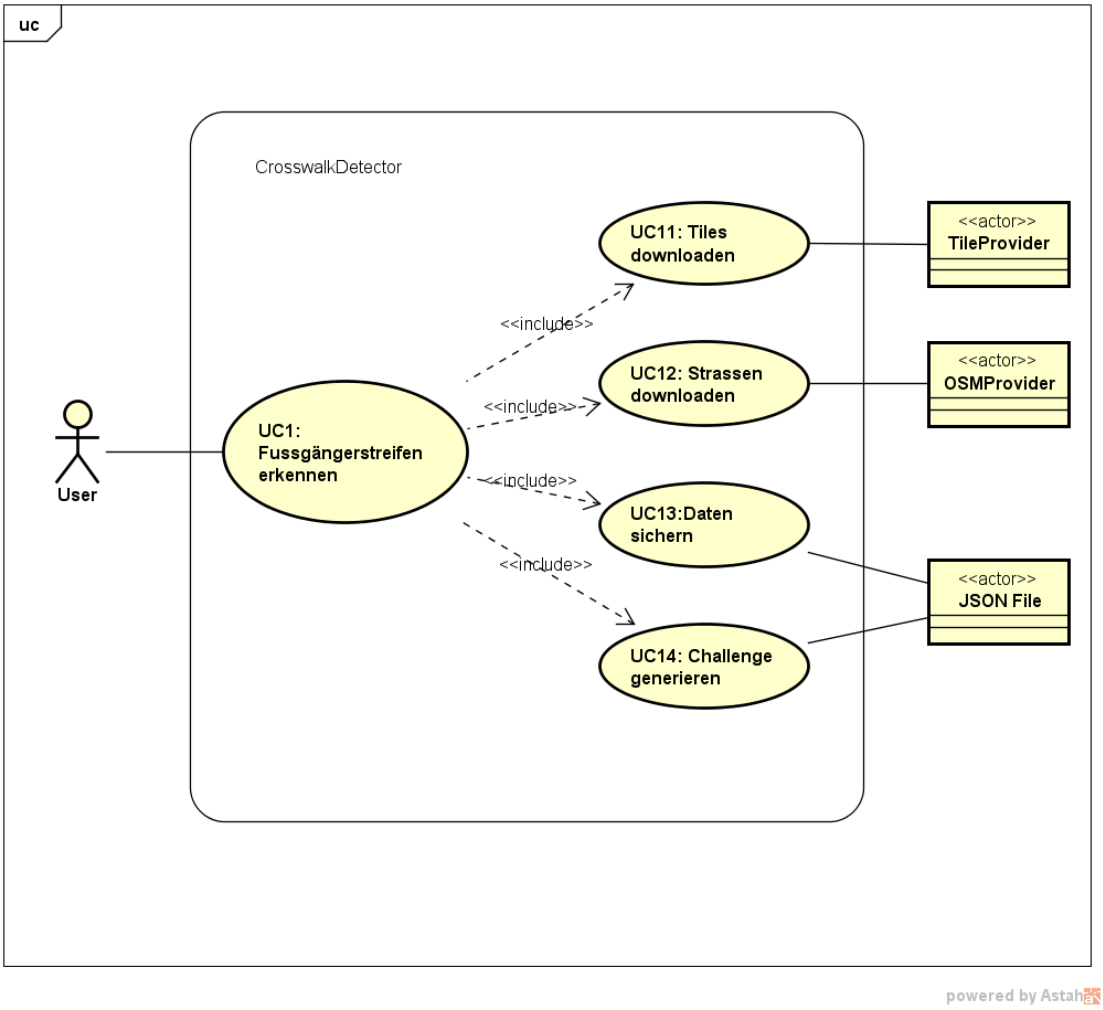

Abbildung 2.1: Use Case Diagramm

Use Cases Brief

UC1: Fussgängerstreifen erkennen Der User startet die Applikation und gibt als Eingabeparameter ein Bounding Box an, welche nach Fussgängerstreifen durchsucht wird. Dabei werden Orthofotos mit Hilfe eines Erkennungsalgortithmus abgearbeitet.

UC11: Tiles downloaden Ein TileProvider stellt Orthofotos zur Verfügung, welche heruntergeladen werden müssen. Diese werden im Anschluss dem Erkennungsalgorithmus zur Verfügung gestellt.

UC12: Stassen downloaden Ein OSMProvider stellt Informationen zu Strassen und Fussgängerstreifen zur Verfügung, welche vom Erkennungsalgorithmus genutzt werden. Mit diesen Daten kann die Suche präzisiert werden, sowie der Download von Orthofotos reduziert werden.

UC13: Daten sichern Die erkannten Fussgängerstreifen werden in einer JSON Datei persistiert. Dabei sind die Positionen (Koordinate lat/lon) relevant.

UC14: Challenge generieren Mit Hilfe der persistieren Daten wird ein Challenge generiert, welche die Daten über eine Crowdsourcing-System in OpenStreetMap integriert.

Use Cases Fully Dressed

Scope	CrosswalkDetection System
Level	User Goal
Primary Actor	User
Stakeholders	• System: Möglichst alle Fussgängerstreifen erkennen • User: Einmal gestartet, läuft alles autonom
Preconditions	• User muss Bounding Box bestimmen • OSMProvider muss verfügbar sein • TileProvider muss verfügbar sein
Postconditions	Koordinaten der Fussgängerstreifen persistiert
Main Success Scenario	1. CrosswalkDetection wird mit Angabe der Boundingbox aufgerufen 2. Daten von OSMProvider werden heruntergeladen 3. Orthofotos von TileProvider werden heruntergeladen 4. Erkennungsalgorithums erfasst Fussgängerstreifen 5. Daten sind persistiert
Extensions	1. a) Bounding Box wird aufgeteilt für Parallelisierung 2. b) Nur Informationen für Strassen und Fussgängerstreifen sind relevant 3. b) Nur Orthofotos, welche Strassen beinhalten, werden heruntergeladen.
Special Requirements	Benutzer soll gut geführt werden und bei Unklarheiten von Parametern Informationen erhalten.
Frequency of Occurrence	Der Vorgang darf beliebig oft wiederholt werden.
Open Issues	Falls ein Unterbruch statt findet, soll von diesem Zustand weiter gearbeitet werden.

Tabelle 2.2: Use Case Fully Dressed

2.2.2 Sequenzdiagramm

Queueing

Das Queueing Sequenzdiagramm beschreibt den Parallelisierungprozess.

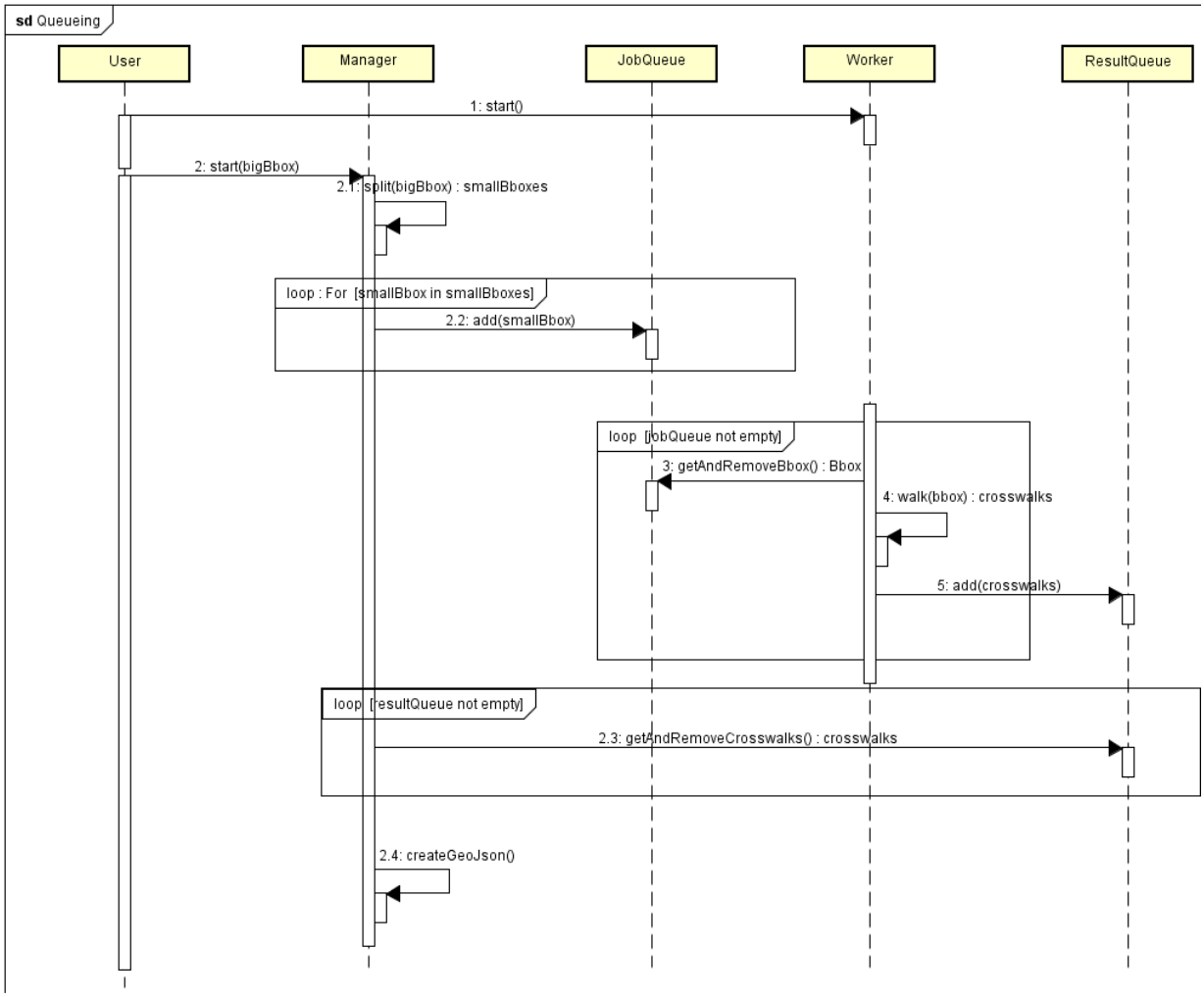

Abbildung 2.2: Sequenzdiagramm Queueing

Walking

Das Walking Sequenzdiagramm beschreibt den Ablauf, wie eine Bounding Box von einem Jobworker abgearbeitet wird.

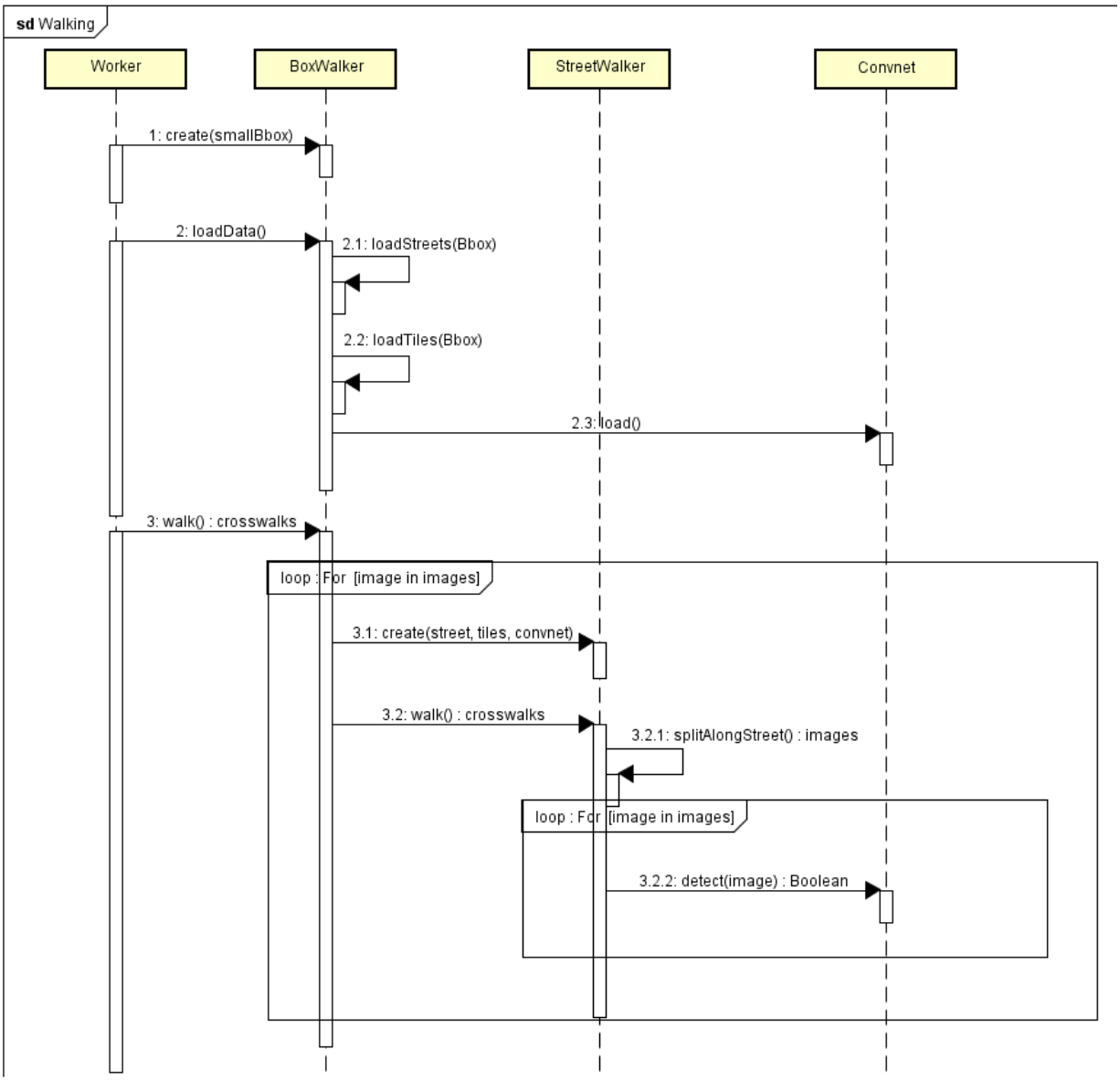

Abbildung 2.3: Sequenzdiagramm Walking

2.2.3 Nichtfunktionale Anforderungen

Funktionalität

Sicherheit Sicherheitsaspekte müssen nicht beachtet werden, es wird nicht mit Personen- oder stark schützenswerten Daten gearbeitet. Der Sourcecode steht unter der MIT Lizenz und ist auf Github verfügbar. Weiter werden die gesammelten Daten über OpenStreetMap für jedermann zugänglich.

Interoperabilität Das System ist auf Orthofotos, sowie Strassen- und Füssgängerinformationen angewiesen. Dazu stehen folgende API zur Auswahl:

- Bing Static Map Data
- Google Static Map API
- MapQuest API
- Overpass

Richtigkeit Die Richtigkeit der erkannten Fussgängerstreifen wird mit Hilfe eines Crowdsourcing-Systems sichergestellt. Dabei verifizieren Freiwillige die erkannten Fussgängerstreifen.

Zuverlässigkeit

Wiederherstellbarkeit Nach einem Systemabsturz oder Stopp der Anwendung, soll die Anwendung ohne Komplikationen wieder gestartet werden können. Beim Neustart soll ab der Absturzstelle weitergearbeitet werden können, ohne das Daten oder bis anhin erbrachte Rechenleistungen verloren gehen.

Fehlertoleranz Fehler in einzelnen Jobs sollen keine Systemweiten Auswirkungen haben. Jede Operation soll im Fehlerfall wiederholt werden können.

Availability Bei Nichtverfügbarkeit des Systems entsteht kein direkter finanzieller Schaden, deshalb ist die Systemverfügbarkeit nicht von oberster Priorität.

Benutzbarkeit

Die Benutzung der Anwendung beschränkt sich auf die Eingabe der Bounding Box für den Bereich an dem Fussgängerstreifen erkannt werden sollen. Ansonsten soll keine Interaktion mit dem Benutzer statt finden. Auf eine grafische Oberfläche wird verzichtet, es ist ein reine Konsolenapplikation.

Robustheit Die Eingabe der Bounding Box durch den Benutzer muss auf Korrektheit überprüft werden. Da die Applikation sehr rechenintensiv ist, soll bei einem Absturz, an der Absturzstelle weiter gearbeitet werden können.

Effizienz

Eine Erkennungsrate von 80% wird angestrebt.
Der Erkennungsprozess soll maximal zwei Wochen dauern für die ganze Schweiz.

Supportability

Internationalization Das System sollte nicht verschiedene Sprachen unterstützen. Die Standardsprache ist Englisch.

2.2.4 Analyse

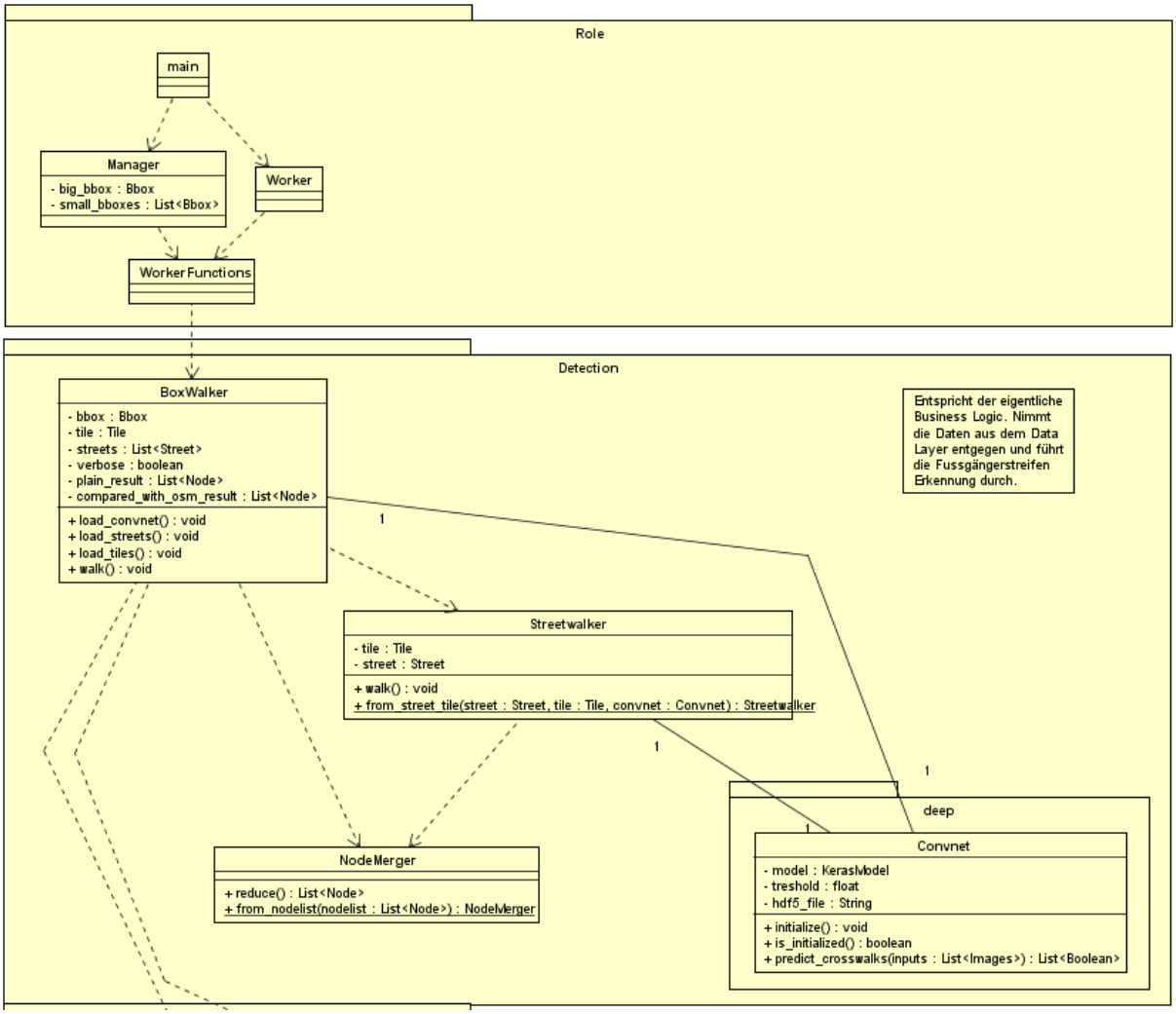

Abbildung 2.4: Domain Modell Teil 1

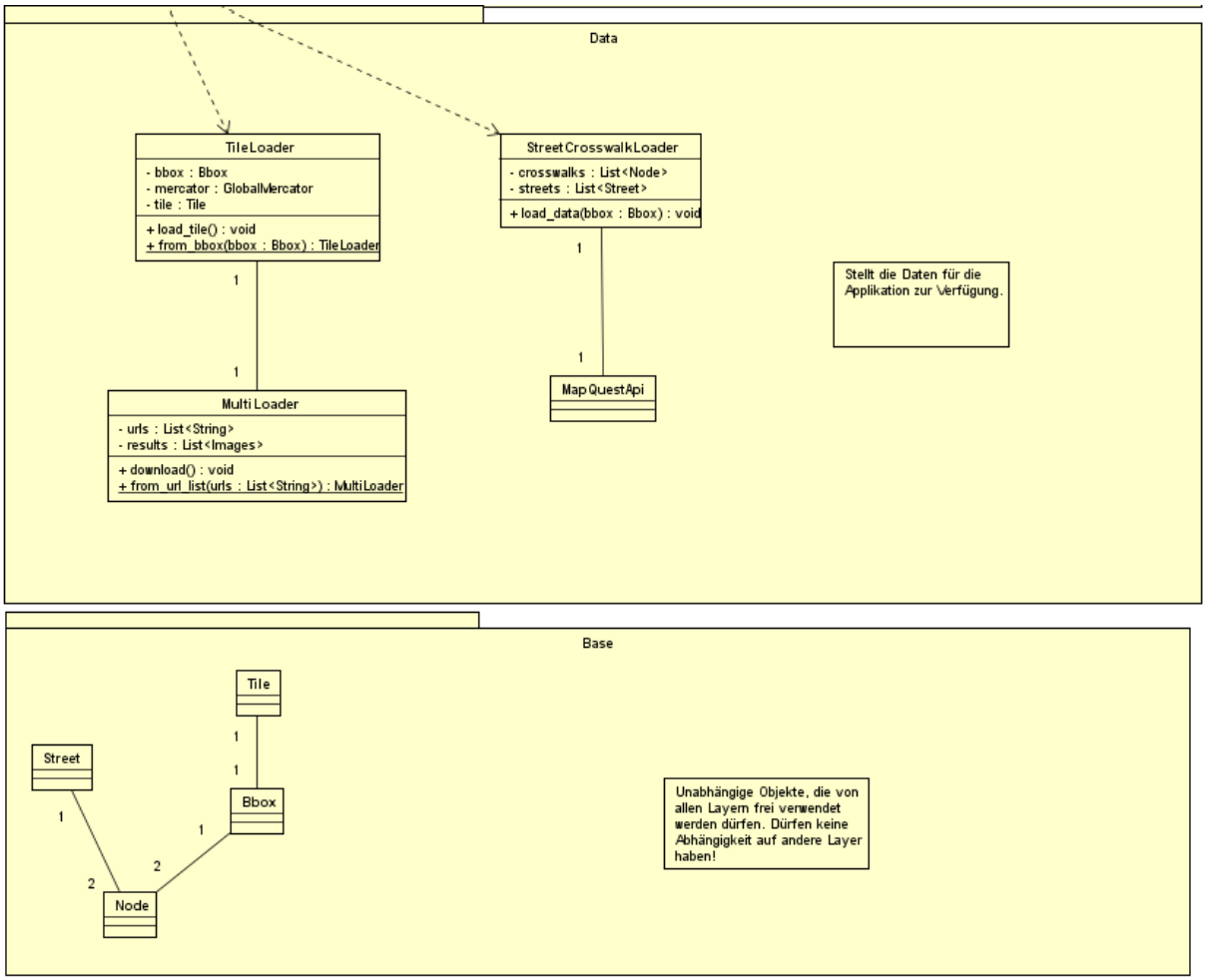

Abbildung 2.5: Domain Modell Teil 2

Beschreibung Domain Modell

Das Domain Modell besteht aus 4 Layern welche von oben nach unten auf einander aufbauen. Die Packages im Projekt sind gleich benannt wie die Layer. Die genauen Funktionen der entsprechenden Klassen sind im Kapitel Implementation einzusehen.

1. Role - Parallelisierung
2. Detection - Bilderkennung
3. Data - Dataprovider
4. Base - Grundelemente

Base

Der Base Layer stellt Grundklassen wie ein Node oder eine Street zur Verfügung. Dieser Layer weisst nur Abhängigkeiten zu verwendeten Bibliotheken auf.

Data

Der Data Layer stellt den klassichen Data Link Layer dar. Er stellt Klassen zur Verfügung, um auf die entsprechenden Daten zuzugreifen. In unserem Fall sind es Orthofotos und OpenStreetMap Daten. Dieses Package benutzt den Base Layer.

Detection

Der Detection Layer beinhaltet die Funktionen zur eigentliche Bilderkennung. Dieser Layer holt seine Daten aus dem Data Layer und benutzt den Base Layer.

Role

Der Role Layer managt die Parallelisierung. Er greift ausschliesslich auf den Detection und auf den Base Layer zu.

Entscheid 4. Main Datei

Die Main Datei ist hier speziell zu erwähnen. Main verwaltet die Kommandozeilenaufrufe und startet das Programm dementsprechend. Diese Datei sollte eigentlich in einem eigenen Presentation Layer sein. Da wir jedoch keinen separaten Layer nur für eine Klasse machen, haben wir Main in den Role Layer verschoben.

2.3 Implementation

Der Implementationsteil ist wie das Projekt selbst in 3 Teile aufgesplittet.

1. Data Access - data
2. Bilderkennung - detection
3. Parallelisierung - role

2.3.1 Data Layer

Der Data Accss Layer regelt den Zugriff auf die Orthofotos mit Bing Maps, sowie die Strassen und Fussgängerstreifen, welche mit Hilfe von OpenStreetMap Daten über die Mapquest API zur Verfügung gestellt werden. Die Daten werden von den entsprechenden Quellen heruntergeladen und für die Detektion in ein passendes Format aufbereitet.

Mapquest

Mapquest [Inc15a] wird in diesem Projekt als Schnittstelle zu den OpenStreetMap Daten verwendet. Dazu bieten sich die Developer Accounts an, welche auf 15000 Abfragen pro Monat begrenzt sind, was unseren Abfrageumfang ausreichend abdeckt. Folgende Daten benötigen wir von der API:

Strassen:	Der Suchalgorithmus folgen den Strassenverläufen, was die Suche effizienter und einfacher werden lässt.
Fussgängerstreifen:	Schon erfasste Fussgängerstreifen werden mit den von uns Entdeckten verglichen und nur diejenigen, welche noch nicht in erfasst sind, werden abgespeichert.

Application Key

In der Tabelle ist der Application Key aufgeführt, der für das Projekt Crosswalk Deteciton eingesetzt wurde.

Consumer Key	YKqJ7JffQIBKyTgALLNXLVrDSaiQGtiI
Consumer Secret	3DO1eoLMxSqPH7Gk
Key Issued	Fri, 09/25/2015 - 07:17
Key Expires	Never

Tabelle 2.3: MapQuest Application Key

Beispiel Abfragen Um den Entwicklern beim Erstellen der Abfragen zu unterstützen wird auf Webseite [Inc15b] ein Xapi Service Developer's Guide zur Verfügung gestellt.

HTTP Request Bounding Box: 47.367,8.545,47.367,8.544 (Rapperswil)

- `http://open.mapquestapi.com/xapi/api/0.6/node[highway=*][bbox=8.544,47.367,8.545,47.367]?key=YKqJ7JffQIBKyTgALLNXLVrDSaiQGtiI`

XML File Response Als Antwort auf den HTTP Request gibt es ein XML File, welches die Strassen in Form von Nodes beinhaltet.

```
<osm xmlns:xapi="http://jxapi.openstreetmap.org/">
<node id="32860913" version="8"
timestamp="2015-08-06T15:21:13Z" uid="6087"
lat="47.2254172" lon="8.8175171">
<tag k="highway" v="traffic_signals"/>
</node>
</osm>
```

Abfrage Um das Resultat einer Abfrage einzugrenzen, bietet die API diverse Parameter und Tags die angegeben werden können. Damit wir nicht zu viele API Requests haben, können wir eine **Bounding Box**, sowie den Tag **highway=*** verwendet werden. Damit ist nur noch eine Abfrage pro Bounding Box (ca. 2 auf 2 Kilometer) nötig. Im Code sieht dies folgendermassen aus:

Python Request Abfrage mit Verwendung der httplib2 Library.

```
import httplib2

url =  'http://open.mapquestapi.com/xapi/api/0.6/node
                [highway=*][bbox=8.544,47.367,8.545,47.367]?
                key=YKqJ7JffQIBKyTgALLNXLVrDSaiQGtiI}'
resp, content = httplib2.Http().request(url)
```

Das Resultat der Abfrage ist im XML Format, Python bietet für die Verarbeitung von XML Daten die Library **ElementTree**.
In einem nächsten Schritt schränken wir das Resultat weiter ein. Denn nicht auf allen Strassen sind Fussgängerstreifen möglich. Für uns relevant sind:

- road, trunk, primary, secondary, tertiary, unclassified, residential, service, trunk_link, primary_link, secondary_link, tertiary_link, pedestrian

Am Ende der Verarbeitung resultiert eine Liste aller relevanten Strassen und alle Fussgängerstreifen.

Bing Maps

Die wichtigsten Daten für die Suche sind die Orthofotos, welche wir über Bing Maps beschaffen. Dabei hilft uns das Python Skript globalmaptiles.py, welches den Umgang mit dem QuadTree Format von Bing Maps und das Umrechnen der Koordinaten zu den entsprechenden Tiles stark vereinfacht. Das Herunterladen der Bilder lässt sich parallelisieren, was die Performance enorm steigert. In Python kann dazu die Library **multiprocessing.dummy import Pool as ThreadPool** verwendet werden.

Beispiel Download Im Nachfolgenden Code ist gezeigt, wie der parallele Download implementiert wird.

```
from multiprocessing.dummy import Pool as ThreadPool
from PIL import Image
import urllib2
import StringIO

def start_download(urls):
     pool = ThreadPool(self.nb_threads)
     images = pool.map(download_image, urls)
     pool.close()
     pool.join()
     return images

def download_image(url):
    request = urllib2.Request(url)
    response = urllib2.urlopen(request)
    content = response.read()
    image = Image.open(StringIO.StringIO(content))
    return image
```

Das Resultat des Downloads ist eine Liste mit den Orthofotos, welche im Anschluss von dem Suchalgorithmus weiter verwendet werden.

Tiles à la Google Maps

Maptiler bietet auf ihrer Webseite [Př15] ein Python Skript an, welches den Umgang mit Tiles stark vereinfacht. Unter anderem wird die Umrechnung von Meter zu Latitude/Longitude, Meter zu Pixel und Tiles zu QuadTree [Mic15b] (Bing Format für Tiles) zur Verfügung gestellt.

Abbildung 2.6: Tiles à la Google Maps

QuadKey

Die Quadkeys von Bing Maps bauen sich wie auf der Abbildung ersichtlich auf. Jedes Zoomlevel führt dazu, dass der QuadKey um eine Stelle zu nimmt. Somit kann die Anzahl der Tiles für das entsprechende Zoomlevel mit dem Zoomlevel als Exponenten zur Basis 4 angegeben werden ($4^{Zoomlevel}$).

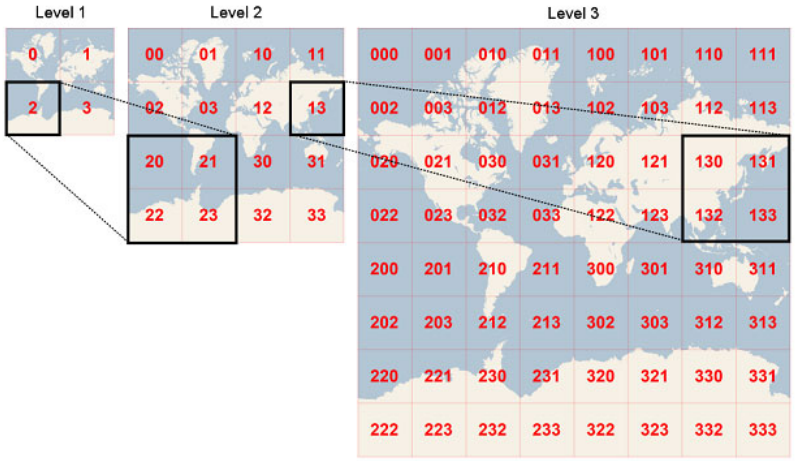

Abbildung 2.7: QuadTree

Beispiel Code

Das folgende Beispiel zeigt die Umrechung vom WGS84 Koordinatensystem zu Meter, zu Tiles und schlussendlich zum Quadkey. (Auf dem Zoomlevel 19)

```
from src.data.globalmaptiles import GlobalMercator

zoom = 19
latitude = 47.0
longitude = 8.0
mercator = GlobalMercator()

meter_x, meter_y = mercator.LatLonToMeters(latitude, longitude)
tile_x, tile_y = self._mercator.MetersToTile(meter_x, meter_y, zoom)
quadkey = mercator.QuadTree(tile_x, tile_y, zoom)
```

2.3.2 Detection Layer

Anfertigung der Bilder entlang der Strassen

Damit das Convnet die Einteilung zwischen Fussgängerstreifen und Nicht-Fussgängerstreifen machen kann, braucht es ein RBG Bild mit 50 x 50 Pixeln als Input. Diese Inputbilder werden aus dem gedownloadetem Orthofoto ausgeschnitten. Mithilfe der Strassenkoordinaten muss nicht das ganze Orthofoto in kleine Bilder aufgeteilt, sondern die Inputbilder können nur entlang der Strassen ausgeschnitten werden. Der Abstand zwischen zwei Inputbildern (auch Schrittweite) ist so gewählt, das eine gewisse Überlappung vorliegt.

Entscheid 5. Eckdaten Anfertigung Inputbilder

Die Inputbilder für das Convnet sind 50 x 50 Pixel gross, da auf Zoomstufe 19 50 Pixel etwa der Breite von zwei Strassen entspricht. Die Schrittweite wurde so angepasst, dass auch Fussgängerstreifen, die zwischen zwei Inputbildern liegen, erkannt werden.

Abbildung 2.8: Blau: Strassenverlauf, Rot: Ausgeschnittene Bilder 50 x 50 Pixel. Diese Visualisierung der Inputbilder kann für jede beliebige Bounding Box selbst durchgeführt werden. Siehe dazu examples/VisualizeSquaredImages.py im Projektordner.

Convnet

Wir verwenden ein Convolutional Neuronal Network (Convnet) um die automatische Einteilung von Fussgängerstreifen und Nicht-Fussgängerstreifen zu machen. Um ein Convnet verwenden zu können, muss man es zuerst mit entsprechendem Bildmaterial trainieren. Wir sprechen hier über Inputbilder, welche zum Teil automatisch mithilfe der OpenStreetMap Datenbank generiert werden konnten, aber meistens von Hand erstellt werden mussten. Wir haben uns während dieses Projektes ein Datenset aus über 4'400 Fussgängerstreifen- und 32'000 Nicht-Fussgängerstreifenbilder erarbeitet. Diese Arbeit beinhaltete stundenlanges Aussortieren von Bildern.

Abbildung 2.9: 6 x 2 Beispiele für Fussgängerstreifen. Wie auf den Bildern zu sehen ist, verdecken immer wieder Gegenstände das gelbe Muster. Die unterschiedliche Bildqualität der Orthofotos behinderte das Erkennen stark.

Abbildung 2.10: 6 x 2 Beispiele für Nicht-Fussgängerstreifen. Das Nicht-Fussgängerstreifen Set beinhaltet alle möglichen Strassensituationen von Stadt bis Land. Besonders bei gelben Autos oder Strassenmarkierungen hat das Convnet mühe.

Ein Beispiel für ein Training eines Convnets kann unter der Datei examples/ConvnetTrainer.py eingesehen werden. Die detaillierte Beschreibung der Funktionsweise eines Neuronalen Netzes würde den Rahmen dieser Arbeit sprengen. Eine genaue Beschreibung der Abläufe kann aus den Vorlesungsunterlagen von Stanford [Kar15] genommen werden. Das in dieser Arbeit benutzte Netz ist unter dem Namen VGGNet (Karen Simonyan, Andrew Zisserman 2014)[KS15] bekannt. Es wurde Ende 2014 veröffentlicht und ist seit diesem Zeitpunkt eines der besten Neuronalen Netze für Bilderkennung. Ein mithilfe der Library Keras trainiertes Neuronales Netz kann in eine HDF5 Datei abgespeichert werden. Nach dem Laden der Datei kann man das trainierte Netz dann immer wieder verwenden.

Convnet Klasse

Die Klasse src/detection/deep/Convnet.py schliesst das neuronale Netz in eine einfach zu verwendende Klasse ein.

```
from src.detection.deep.Convnet import Convnet

1. convnet = Convnet.from_verbose(verbose=True)
2. convnet.initialize()
3. convnet.treshold = 0.7

4. result_list = convnet.predict_crosswalks(pil_image_list)
```

1. Konstruiert die Convnet Klasse mit einer Factory
2. Lädt das gespeichert Neuronale Netz mithilfe der HDF5 Datei. Keras übersetzt das Netz in diesem Schritt in C++ und kompiliert es. Die Ausführung ist dementsprechend schnell.
3. Manuelles Setzen eines Schwellwerts. Der Schwellwert muss zwischen 0 und 1 liegen. Standartmässig ist er auf 0.9 gesetzt. Somit muss das Netz zu 90% sicher sein, dass es ein Fussgängerstreifen ist. Dieser Wert ergab in unseren Experimenten die besten Resultate.

4. Kategorisiert eine Liste von 50 x 50 Pixel RGB PIL Bilder. Diese Funktion nimmt eine Liste entgegen, da Keras die Bilder so parallel auf mehreren CPU Kernen verarbeiten kann. Als Rückgabewert erhält result_list eine Liste von Booleans.

Fussgängersteifenerkennung entlang der Strassen

In diesem Abschnitt kommen die Strasseninformationen, Orthofotos, Inputbilder und das Convnet zusammen. Die Klassen StreetWalker und BoxWalker stellen Funktionen zur Verfügung, mithilfe derer alle Strassen innerhalb einer Bounding Box entlang "gelaufen" und die Fussgängerstreifen erkannt werden.

StreetWalker

Der StreetWalker löst das Erkennungsproblem entlang eines Strassenabschnitts. Ein Strassenabschnitt besteht aus zwei Koordinaten. Der Walker geht von der ersten Koordinate auf die zweite Koordinate zu entlang einer geraden Linie. Zum Schluss fasst er doppelt gefundene Fussgängerstreifen mithilfe des NodeMergers zusammen (siehe unten).

```
from src.detection.StreetWalker import StreetWalker

1. walker = StreetWalker.from_street_tile(street, tile, convnet)
2. crosswalks = walker.walk()
```

1. Erstelle einen StreetWalker anhand eines Strassenabschnitts, des Orthofotos und des Convnets.
2. Walker läuft entlang dieses Strassenabschnitts und gibt eine Liste von Koordinaten der gefundenen Fussgängerstreifen zurück.

BoxWalker

Der BoxWalker löst das Erkennungsproblem für eine ganze Bounding Box. Zuerst lädt er die Orthofotos, die Strassen- und Fussgängerstreifeninformationen herunter. Danach erfolgt der eigentliche Erkennungsprozess, indem er für jeden Strassenabschnitt einen Streetwalker initialisiert. Zu guter Letzt vergleicht er die gefundenen Fussgängerstreifen mit den bestehenden aus OpenStreetMap und lässt den NodeMerger doppelt gefundene Fussgängerstreifen zusammenfassen (siehe unten).

```
from src.detection.BoxWalker import BoxWalker

1. rapperswil = Bbox.from_bltr(47.224553, 8.816052, 47.227839, 8.820165)

2. walker = BoxWalker(rapperswil)
3. walker.load_convnet()
4. walker.load_tiles()
5. walker.load_streets()

6. walker.walk()

7. all_crosswalk_nodes = walker.plain_result
8. compared_crosswalk_nodes = walker.compared_with_osm_result
```

1. Erstelle eine Bounding Box anhand von Koordinaten.
2. Erstelle einen BoxWalker mit der Bounding Box.
3. Lade das Convnet.
4. Lade die Orthofotos.
5. Lade die Strasseninformationen.
6. Eigentlicher Erkennungsprozess.
7. In plain_result sind nun alle gefundenen Fussgängerstreifen enthalten.
8. In compared_with_osm_result sind nur noch die Fussgängerstreifen enthalten, die nicht schon in OpenStreetMap vorhanden sind.

NodeMerger

Die NodeMerger-Klasse versucht das Problem der doppelten Erkennung zu lösen. Dadurch, das sich die Inputbilder überlappen, kann es dazu kommen, das Fussgängerstreifen doppelt erkannt werden. Das gleiche Problem tritt auf, wenn mehrere Strassen über den gleichen Fussgängerstreifen führen.

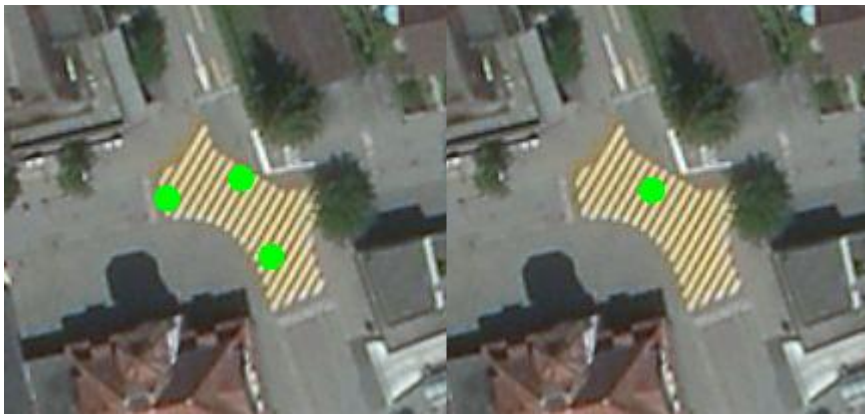

Abbildung 2.11: Beispiel NodeMerger - Dieser Fussgängerstreifen liegt auf einer Kreuzung und ist ziemlich gross. Links vor der Zusammenfassung, rechts danach. Der Merger nimmt den Durchschnitt aller drei Koordinaten als neue Koordinate.

Der NodeMerger nimmt alle gefundenen Koordinaten entgegen und berechnet alle Abstände von gefundenem Fussgängerstreifen zu gefundenem Fussgängerstreifen. Sind die Koordinaten genug nah beieinander, werden sie zusammengefasst. Genau genommen wird hier ein Graph mit Knoten und Kanten aufgebaut. Die Koordinaten sind die Knoten und die Knoten werden verbunden, wenn sie genug nahe beieinander sind. Zum Schluss werden alle Knoten zusammengefasst, welche verbunden sind.

Kritisch ist hier die richtige Distanz zu finden. Zu grosse Distanzen fassen mehrere Fussgängerstreifen zu einem zusammen. Zu kleine Distanzen lassen zu viele Punkte übrig.

Entscheid 6. NodeMerger Distanzen

StreetWalker: 10 Meter, BoxWalker: 7 Meter

Vergleich gefundene Fussgängerstreifen mit bestehenden Fussgängerstreifen in OSM

Der Boxwalker führt nach dem Erkennungsvorgang ein Vergleich mit den bestehenden Fussgängerstreifen in OpenStreetMap durch. Dabei rechnen wir die Distanz von den gefundenen Fussgängerstreifen zu den bestehenden Fussgängerstreifen aus. Ist die Distanz kleiner 5 Meter, so besteht der Fussgängerstreifen schon und wird in walker.compared_with_osm_result nicht aufgelistet.

2.3.3 Role Layer

RQ [Dri15] (Redis Queue) ist eine einfache Library für Python um Jobs in Redis einzureihen. Im Anschluss sind Codebeispiele dazu angeführt.

Beispiel Queue

```
from rq import Queue, use_connection
from redis import Redis

class RQueue:
    def __init__(self):
        redis = Redis(ip, port, password)
        q = Queue(connection=redis)
        job = q.enqueue(self.add, 2, 3)
        print job.result

    def add(self, x, y):
        return x + y
```

Beispiel Worker

```
from rq import Queue, Connection, Worker
from redis import Redis

redis = Redis(ip, port, password)
with Connection(redis):
    qs = map(Queue, sys.argv[1:]) or [Queue()]
    w = Worker(qs)
    w.work()
```

Aufbau Die Applikation beinhalten die Rollen:

- Manger
  - Teilt grosse Boundingbox in kleine auf.
  - Stellt kleine Boundingboxes als Workerjobs in die Queue.
- Jobworker
  - Verarbeitet die Workerjobs.
  - Führt den Erkennungsprozees durch.
  - Stellt die Resultate als Resultjobs in die Queue.
- Resultworker
  - Verarbeite die Resultjobs.
  - Speichert die Resultate in einem JSON File.

2.4 Test

2.4.1 Unittests

In der Python Standard Library gibt es das Unit Testing Framework **unittest**, das es erlaubt Unittests zu implementieren.

Beispiel Test

```
import unittest
import json, os
from src.role.WorkerFunctions import store, PATH_TO_CROSSWALKS
from src.base.Node import Node

class TestWorkerFunctions(unittest.TestCase):

    def setUp(self):
        self.remove_file()

    def tearDown(self):
        self.remove_file()

    def test_store_two_crosswalks(self):
        crosswalks = [Node(47.0, 8.0), Node(47.1, 8.1)]
        store(crosswalks)
        with open(Constants.PATH_TO_CROSSWALKS, 'r') as f:
            data = json.load(f)
        self.assertTrue(len(data['crosswalks']) == 2);

    def remove_file(self):
        if os.path.exists(PATH_TO_CROSSWALKS):
            os.remove(PATH_TO_CROSSWALKS)
```

2.4.2 CircleCI

CircleCI[Cir15] ist ein Tool für Continuous Integration und Deployment, welches wir einsetzen um unsere Tests nach einem Update in unserem Github Repository automatisch durchzuführen. CircleCI ermöglicht eine Anbindung zu einem Docker Image. Damit konnten wir die vielen Dependencies abdecken, welche unsere Applikation beinhaltet.

Kapitel 3

Projektmanagement

3.1 Rollen und Verantwortlichkeiten

3.1.1 Prof. Keller Stefan

Abbildung 3.1: Prof. Stefan Keller

Prof. Stefan Keller, Mitarbeiter des Institut für Software (IFS) und Dozent an der HSR, ist in diesem Projekt als Betreuer tätig.

Somit wird er die Aufsicht, wie auch die Bewertung der Semesterarbeit duchführen.

3.1.2 Bühler Severin

Abbildung 3.2: Severin Bühler

Severin Bühler, Student an der HSR, ist Entwickler des Projektes.

3.1.3 Kurath Samuel

Abbildung 3.3: Samuel Kurath

Samuel Kurath, Student an der HSR, ist Entwickler des Projektes.

3.2 Entwicklungsumgebung und Infrastruktur

3.2.1 IDE (Integrated Development Environment)

Entscheid 7. PyCharm

Beiden Projektmitgliedern ist JetBrains Intellij bekannt und PyCharm ist im Umgang nahe zu identisch. Für Studenten sind die Entwicklungsumgebungen kostenlos verfügbar.

3.2.2 SCM (Source Control Management)

Entscheid 8. GitHub

Der Umgang mit Git ist beiden Projektmitglieder bestens bekannt. GitHub ist ohne Unkosten von überall verfügbar. Das Geometalab der HSR publiziert über diesen Weg diverse Projekte.

3.2.3 CI (Continuous Integration)

Entscheid 9. CircleCI

Das finden eines passenden Continuous Integration Tools stellte sich schwieriger dar, als zu Beginn des Projektes erwartet. Während dem SE2 Projekt haben wir Bekanntschaft mit Travis CI gemacht, welches die vielen Abhängigkeiten unseres Codes nicht abdecken konnte. Mit CircleCI fanden wir eine Lösung, die auf Docker Hub zugreifen kann, dann den Build des Images durchführt und schlussendlich die Test durchführt.

3.2.4 Projektmanagement Tool

Entscheid 10. Jira

Jira ist den Projektmitgliedern schon aus dem SE2-Projekt bekannt und hat sich sehr bewährt. Das Dashboard ist übersichtlich gestaltet. Es ermöglicht eine Übersicht über die aktuellen Tasks auf einen Blick. Alle Mitglieder haben jederzeit Zugriff auf die Plattform, was die Transparenz erhöht. Weiter bietet Jira diverse Reports um Auswertungen über das Projekt zu fahren.

3.3 Planung

Am Anfang es Projektes haben wir eine grobe Planung zusammengestellt. Dabei haben wir die Phasen und Meilensteine definiert. Während dem Projekt stellten wir immer wieder grössere oder kleinere Abweichungen an der zu Beginn definierten Planung fest. Dieses ist jedoch nicht erstaunlich, da nie absolut korrekt geplant werden kann. Um solche Schwierigkeiten zu handhaben, erstellten wir ein Risikomanagementdokument.

3.3.1 Phasen

1. Inception
    (a) Aufgabenstellung ausarbeiten
2. Elaboration1
    (a) Evaluation der Algorithmen (Bilderkennung)
3. Elaboration2
    (a) Prototyp 1 (In Orthofotos, Out Koordinaten)
    (b) Prototyp 2 MapRoulett
4. Construction1
    (a) Schnittstelle Orthofotos
    (b) Optimierungen durch Strassenverlauf und ähnliches
5. Construction2
    (a) MapRoulette (Tags und Quiz)
    (b) Koordinaten erfassen
6. Transition
    (a) Dokumentation abschliessen
    (b) Challenge auf Maproulette

3.3.2 Meilensteine

1. MS1 Algorithmus für Bilderkennung evaluiert
2. MS2 Prototyp erstellt
3. MS3 Automatisierte Datenverarbeitung
4. MS4 Applikation fertiggestellt
5. MS5 Challenge auf Maproulette

3.3.3 Zeitplanung

Aufwand: 14 Wochen zu 2 * 16 Stunden = **448 Stunden**

Inception	1 Woche
Elaboration1	3 Wochen
Elaboration2	4 Wochen
Construction1	3 Wochen
Construction2	1 Wochen
Transition	2 Wochen

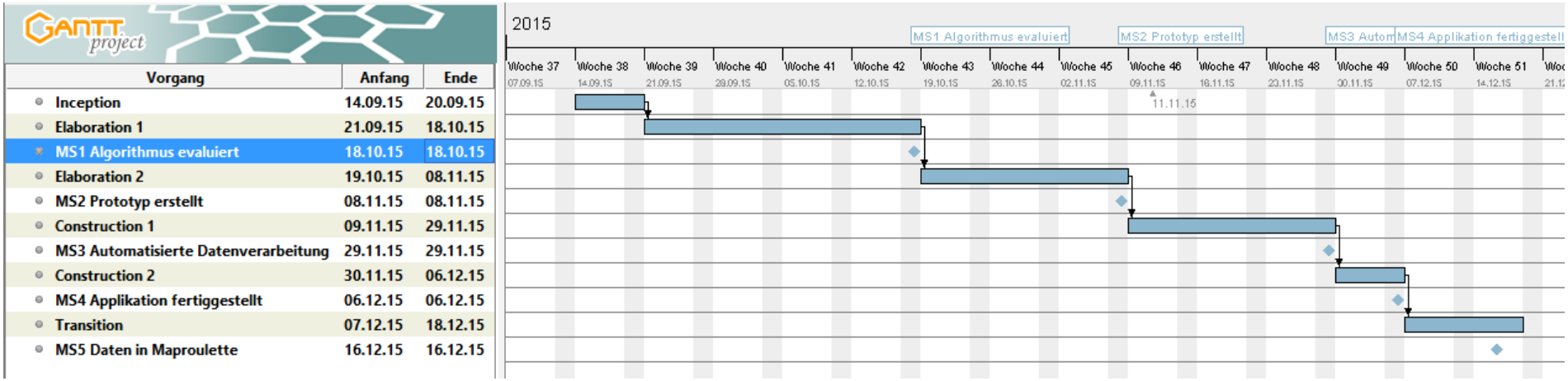

Abbildung 3.4: Gantt Chart

3.4 Risiken

Um den Problemen, die während des Projekts auftreten können entgegenzuwirken, haben wir eine Risiko Analyse durchgeführt. Diese konnte dann bei der Planung eingesetzt werden.

3.4.1 Technische Risiken

Nr	Titel	Beschreibung	maximaler Schaden	Eintrittswahrscheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten
R1	Einarbeitung Python	Python ist den Teammitgliedern teils bekannt, jedoch wurde noch kein grösseres Projekt mit dieser Sprache entwickelt.	40h	10%	4h	Evaluation des Wissensstandes	Informationen bei Studenten einholen, die Python gut kennen
R2	Installation OpenCV	Die Installation von OpenCV mit den Contrib Package ist bekanntermassen ein grosse Hürde	16h	50%	8h	Installation mit Tutorials durchführen	Rücksprache mit Felix Morgner
R3	Detektion	Der Algorithmus, der Fussgängerstreifen erkennt, liefert zu schlechte Resultate und kann nicht gebraucht werden.	40h	50%	20h	Analyse diverser Algorithmen in der Evaluation	Gespräch mit Guido Schuster suchen
R4	Download Orthofoto	Download der Orthofotos von Bing oder ähnlichen Quellen ist nicht möglich	50h	30%	15h	Alternativen im Auge behalten	Auf Bildmaterial der HSR zurückgreifen
R5	Software ist zu langsam	Der Download der Orthofotos oder die Detektion kann einige Zeit in Anspruch nehmen.	60h	70%	42h	Konzept für Parallelisierung erarbeiten	Fläche einschränken, -Grössere und mehrere Maschinen verwenden.

Tabelle 3.1: Risiken - Die technischen Risiken wurden zu Beginn des Projektes, wie in der Tabelle ersichtlich, definiert.

3.4.2 Auswertung

R1 Einarbeitung Python Risiko ist nicht eingetreten, die Entwickler hatte keine Mühe mit Python zu arbeiten.

R2 Installation OpenCV Risiko ist in vollem Umfang eingetreten. Die Installation und Kompilation stellte sich als äusserst Trickreich heraus.

R3 Detektion Die Detektion stellte die Hauptaufgabe unserer Arbeit dar, ist jedoch gleichzeitig eine der Risiko reichsten, da Bilderkennung ein nicht ganz triviales Problem ist. Das Implementieren und Testen der verschieden Algorithmen war sehr zeitaufwändig, was dazu führte, dass auch dieses Risiko eingetroffen ist.

R4 Download Orthofoto Während des Projektes, wechselten wir mehrmals die API für den Download, was sich auch hier auf einen erhöhten Aufwand auswirkte.

R5 Software ist zu langsam Durch den Einsatz von RQ in Kombination mit Redis wurde diesem Risiko Einhalt geboten.

Nr	Titel	Schaden
R1	Einarbeitung Python	0h
R2	Installation OpenCV	20h
R3	Detektion	60h
R4	Download Orthofoto	12h
R5	Software ist zu langsam	0h
Total		92h

Tabelle 3.2: Risikoauswertung

3.5 Soll-Ist-Zeit-Vergleich

Während des ganzen Projektes haben wir in Jira vor jeder Phase die jeweiligen Tasks definiert und den Aufwand dazu geschätzt (Soll). Weiter haben wir dann auch die effektive Zeit zu den Tasks erfasst (Ist).

3.5.1 Inception

Start: 14.09.2015 Ende: 23.09.2015

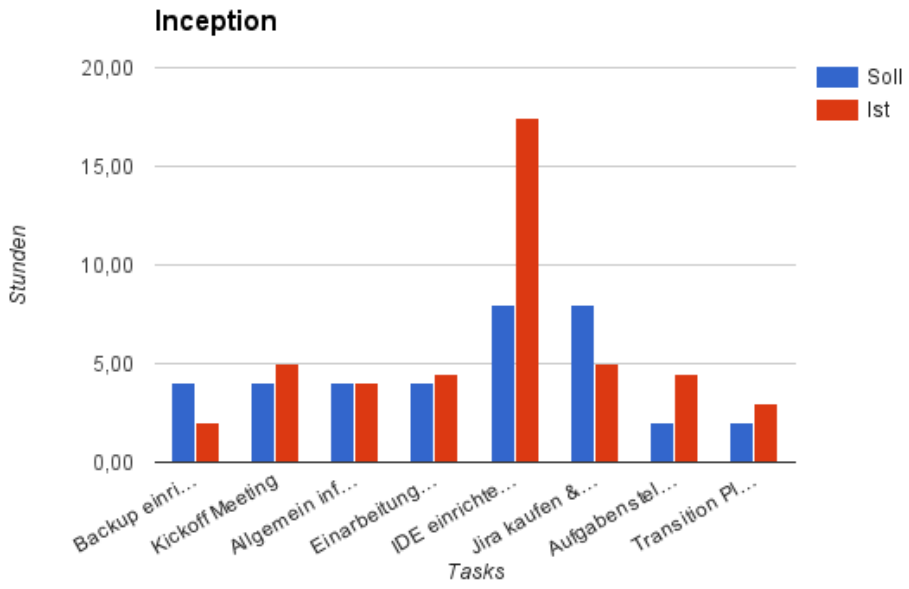

Abbildung 3.5: Inception

3.5.2 Elaboration1

Start: 23.09.2015 Ende: 19.10.2015

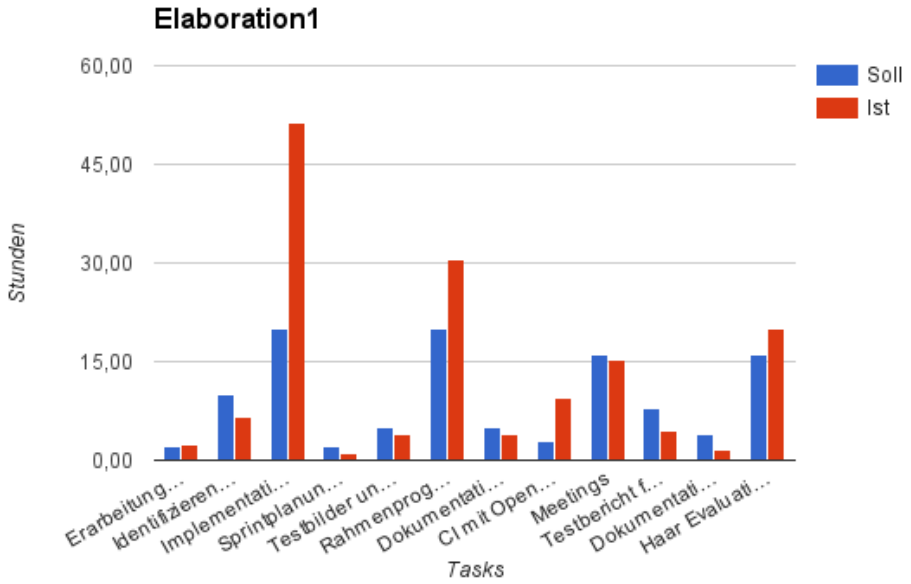

Abbildung 3.6: Elaboration1

3.5.3 Elaboration2

Start: 19.10.2015 Ende: 04.11.2015

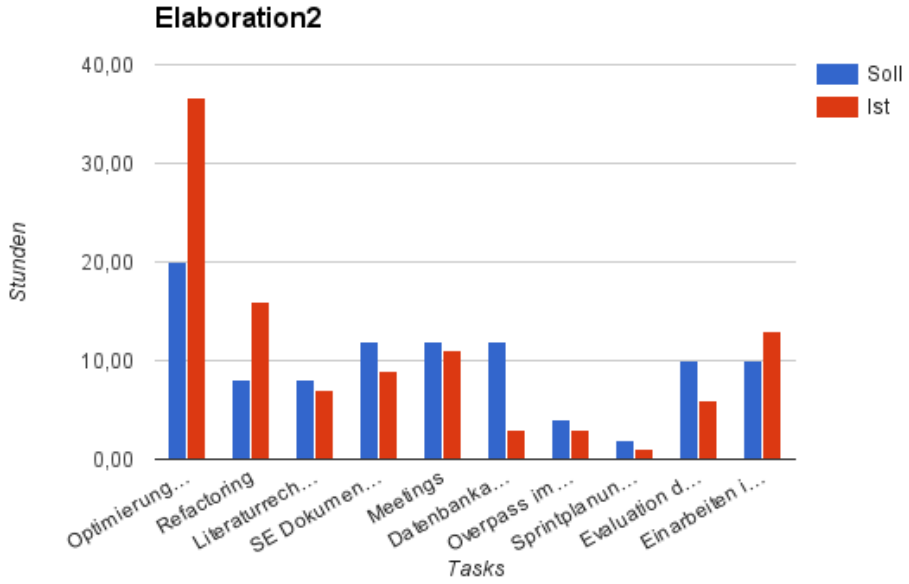

Abbildung 3.7: Elaboration2

3.5.4 Construction1

Start: 04.11.2015 Ende: 25.11.2015

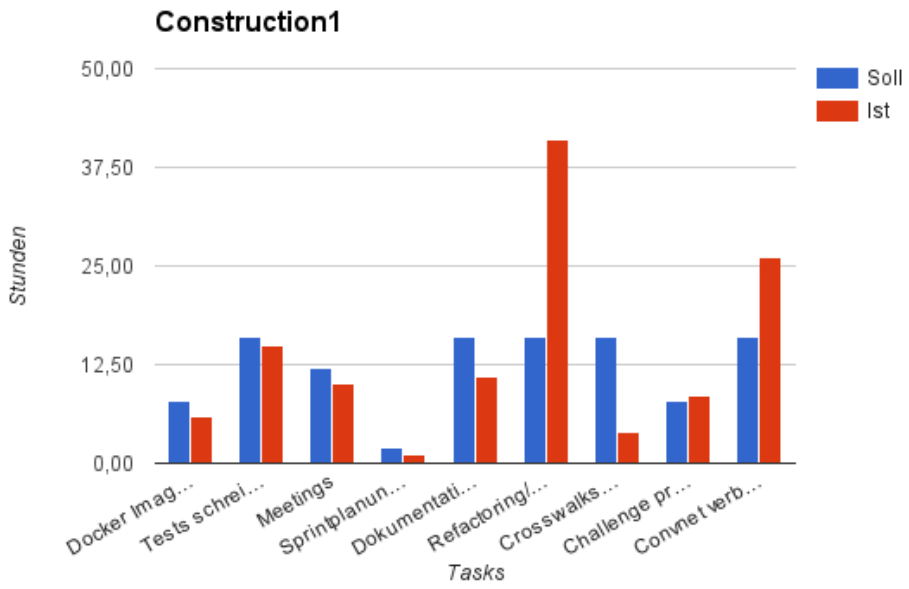

Abbildung 3.8: Construction1

3.5.5 Construction2

Start: 04.11.2015 Ende: 11.11.2015

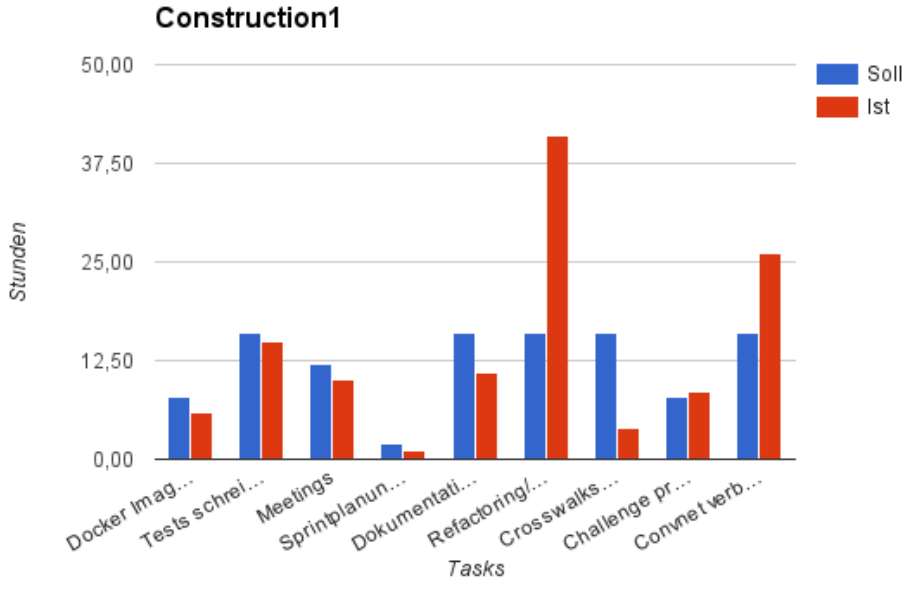

Abbildung 3.9: Construction2

3.5.6 Transition

Start: 11.12.2015 Ende: 18.12.2015

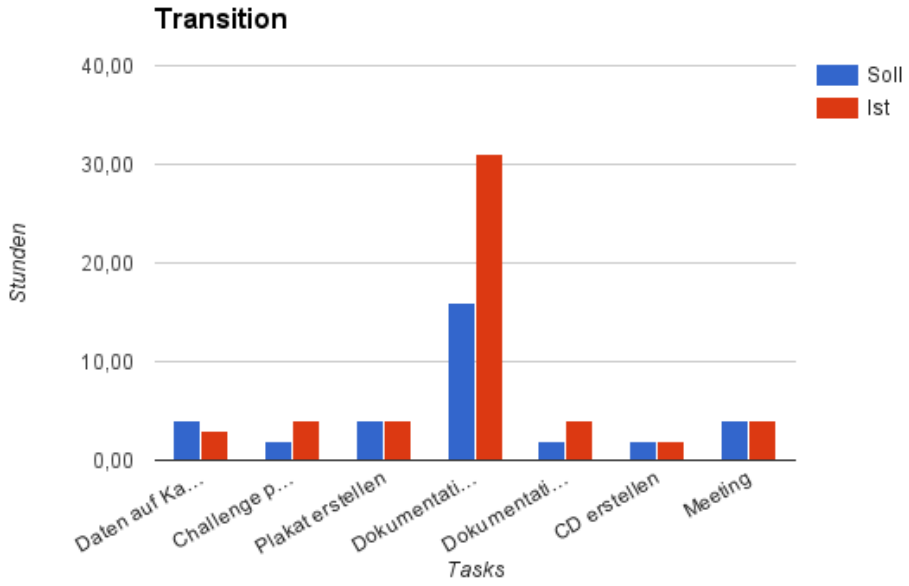

Abbildung 3.10: Transition

3.5.7 Übersicht

Phase	Soll	Ist	Differenz
Inception	36.00	45.50	9.50
Elaboration1	111.00	150.50	39.50
Elaboration2	98.00	105.75	7.75
Construction1	110.00	122.50	12.50
Construction1	58.00	88.50	30.50
Transition	34.00	52.00	18.00
Total	447.00	564.75	117.75

Tabelle 3.3: Phasen

Schätzen ist wie bekannt, ein relativ schwierige Angelegenheit. So haben wir den Aufwand für die jeweiligen Tasks meist zu tief eingestuft. Stark ins Gewicht fiel die Evaluation und Implementierung des Bilderkennungsalgorithmus.

3.6 Codestatistik

3.6.1 Test Coverage

Test Coverage wurde mit dem Tool **nose** durchgeführt.

Datei	Coverage [%]
src/base/Bbox.py	85
src/base/Node.py	97
src/base/Street.py	92
src/base/Tile.py	98
src/base/TileDrawer.py	24
src/data/MapquestApi.py	100
src/data/MultiLoader.py	94
src/data/StreetLoader.py	100
src/data/TileLoader.py	100
src/detection/BoxWalker.py	100
src/detection/NodeMerger.py	89
src/detection/StreetWalker.py	100
src/detection/deep/Convnet.py	97
src/detection/deep/training/Crosswalk_dataset.py	100
src/detection/deep/training.py	100
src/role/Manager.py	91
src/role/WorkerFunctions.py	70
Durchschnitt	90.5

Tabelle 3.4: Test Coverage

3.6.2 Codezeilen

Die Codezeilen wurden mit Hilfe von **CLOC** [AID15] ausgezählt.

Sprache	Dateien	Zeilen
Python	43	2045

Tabelle 3.5: Codezeilen

3.7 Rational Unified Process (RUP)

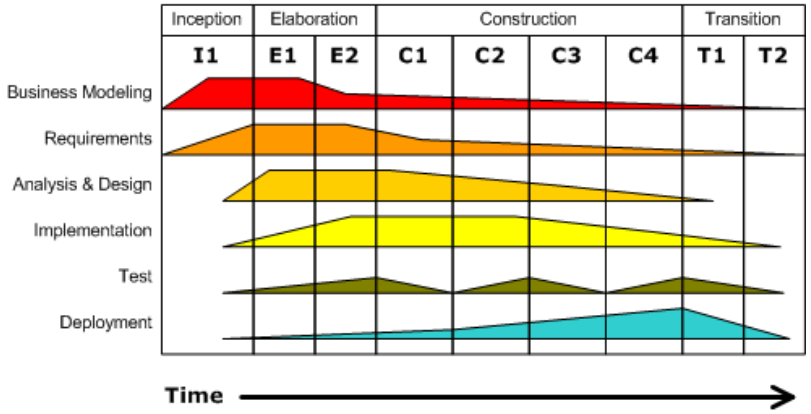

Abbildung 3.11: RUP

Das Projekt Extraction of Crosswalks from Aerial Images führten wir mit dem Vorgehensmodell RUP [Fan15] aus. Dies ist eine iterative Art um Softwareprojekt anzugehen. RUP kennt die Phasen:

- Inception, Elaboration, Construction, Transition

Um eine feinere Aufteilung zu haben, unterteilten wir die Elaboration, sowie die Construction in je zwei Hälften.

Fazit

Wir hatten Mühe unserem Projekt ein RUP Stempel auf zu drücken, da wir bei der Evaluation der Algorithmen immer wieder grosse Teile unserer Lösung überdenken und auf den Kopf stellen mussten. Es wurde viel geschriebener Code wieder über den Haufen geworfen. Weiter war es schwierig den Phasen gerecht zu werden, nur mit vielen Überstunden konnte am Ende der Elaboration ein Prototyp präsentiert werden, der ein angemessenes Resultat lieferte. Im Nachhinein hätte sich ein agilieres Vorgehensmodell wie Scrum oder ähnlich vielleicht besser bewährt.

Kapitel 4

Softwaredokumentation

4.1 Installation

4.1.1 Redis

Die Installation wurde auf einem Ubuntu Server durchgeführt, welcher das Paketverwaltungssystem Advanced Packaging Tool (APT) verwendet. Die im Anschluss aufgeführten Befehle werden in einer Shell ausgeführt. Falls Probleme auftreten, bietet Redis ein Quick Start Dokumentation an.

Installaiton

```
# sudo apt-get install redis-server
```

Konfiguration

```
# redis-cli -p 40001
# CONFIG SET requirepass "crosswalks"
# redis-cli AUTH crosswalks
```

Starten

```
# redis-server --port 40001
```

4.1.2 Keras

Keras ist die Bibliothek, mit der das neuronale Netz betrieben wird. Die Installation hier muss nur durchgeführt werden, um das Projekt weiterzuentwickeln.

Info: Der Stand von Keras, der wir während dieser Arbeit verwendet haben, ist auf der CD zu finden.

Installation Theano

```
# apt-get update
# apt-get install -y git libopenblas-dev python-dev python-pip
python-nose python-numpy python-scipy
# pip install git+git://github.com/Theano/Theano.git
```

Installation Keras

```
# apt-get install -y libhdf5-dev python-h5py python-yaml
# pip install --upgrade six
# cd /root
# git clone https://github.com/fchollet/keras.git
# cd keras
# python setup.py install
```

4.1.3 Docker

Die Installation unserer Applikation beinhaltet diverse Abhängigkeiten, welche für die Installation einerseits viel Zeit in Anspruch nehmen und anderseits auch nicht wirklich trivial sind..

Entscheid 11. Docker

Deshalb haben wir ein Docker Image ersellt, das auf Dockerhub [Kur15] frei zur Verfügung gestellt wird und somit den DevOps Prozess massiv vereinfach.

Installation

Bei der Installation von Docker ist zu beachten, dass die Anwendung nur auf 64-Bit Maschinen läuft. Weiter wurden die nachfolgenden Befehle auf Ubuntu durchgeführt und variieren deshalb je nach Betriebssystemen.

Vorarbeit

```
# sudo apt-key adv --keyserver hkp://p80.pool.sks
-keyservers.net:80 --recv-
keys 58118E89F3A912897C070ADBF76221572C52609D
# echo 'deb https://apt.dockerproject.org/repo ubuntu-precise main'
>> /etc/apt/sources.list.d/docker.list
# apt-get update
# apt-cache policy docker-engine
# sudo apt-get install linux-image-extra-$(uname -r)
```

Docker installieren

```
# sudo apt-get install docker-engine
```

Starten

```
# sudo service docker start
# sudo docker run hello-world
```

4.2 Benutzerhandbuch

4.2.1 Suche der Fussgängerstreifen

Um die Suche der Fussgängerstreifen durchzuführen, muss eine Redis Datenbank zur Verfügung stehen. Weiter muss auf den Rechnern, die als Jobworker tätig sind, Docker installiert sein.
Die Installationen sind in folgenden Abschnitten aufgeführt:

- Redis: Abschnitt 4.1.1 auf der Seite 73
- Docker: Abschnitt 4.1.3 auf der Seite 74

Einführung

Wir haben unsere Applikation in drei Rollen aufgeteilt:

- Manager
  - Unterteilt eine grosse Bounding Box in kleinere Boxen mit einer Höhe und Breite von jeweils 2 Kilometern und stellt dies als Jobs in die Queue.
- Jobworker
  - Arbeite die Jobs der Queue ab.
  - Gefundene Fussgängerstreifen , welche noch nicht in OpenSteetMap erfasst sind, werden als Job Resultat in die Queue gestellt.
- Resultworker
  - Schlussendlich werden die Resultate zusammen getragen und in ein JSON File geschrieben.

Dieser Ablauf ist genauer beschrieben unter dem Abschnitt 3 auf der Seite 28

Anwendung

Dank Docker kann unsere Applikation innert Minuten gestartet werden.

Docker Pull

```
# docker pull murthy10/osm-crosswalk-detection
```

Manager

```
# docker run murthy10/osm-crosswalk-detection REDIS_IP_ADDR
--role manager left bottom right top
```

Left, Bottom, Right, Top entsprechen den Koordinaten im WGS84 Format.
Ostschweiz: left=8.361002, bottom=47.166994, right=8.977610, top=47.706676

Jobworker

```
# docker run murthy10/osm-crosswalk-detection REDIS_IP_ADDR
--role jobworker
```

Jobworker können auf beliebig vielen Rechnern gestartet werden.

Resultworker

```
# docker run murthy10/osm-crosswalk-detection REDIS_IP_ADDR
--role resultworker
```

Die Resultate werden in der Datei crosswalks.json gespeichert. Diese findet man im Verzeichnis in dem der Resultworker gestartet wurde.

Struktur JSON

Die Struktur der crosswalks.json Datei ist folgendermassen aufgebaut:

```
{
        "crosswalks":
        [
                {"latitude": 47.0, "longitude": 8.3},
                {"latitude": 48.0, "longitude": 8.4}
        ]
}
```

4.2.2 Daten visualisieren

Um das Resultat des Erkennungsalgorithmus zu visualisieren bot sich das Tool CartoDB [Car15] an. Dieses ermöglicht Daten in diversen Formaten hochzuladen und auf einer Karte anzuzeigen.

Vorgehen

1. Daten (crosswalk.json) mit den Spalten latitude und longitude. in CSV Format umwandeln
2. CSV Datei in CartoDB als neues Dataset hochladen.

Struktur CSV

Die Struktur der CSV Datei gliedert sich wie folgt:

```
        latitude,       longitude
        47.0,           8.3
        48.0,           8.4
```

Daten selektieren

CartoDB ermöglicht eine Selektion der Daten. So kann zum Beispiel ein Polygon selektiert werden.

```
SELECT * FROM dataset WHERE ST_WITHIN(
        ST_Transform(the_geom, 4326), ST_SetSRID(
        ST_GeomFromGeoJSON('{ "type": "Polygon",
        "coordinates": [
          [ [8.81429672241211,
             47.22971054221563],
            [8.8385009765625,
             47.22877798599878],
            [8.820991516113281,
             47.2185187846731],
            [8.81429672241211,
             47.22971054221563]
          ]
        ] }'),
   4326))
```

4.2.3 Challenge erstellen

Nach dem die Fussgängerstreifen detektiert wurden und die Datei crosswalks.json erstellt wurde, muss dies noch in ein passendes Format für MapRoulette gebracht werden. Dazu haben wir ein Python Skript geschrieben, welches aus jedem gefundenen Fussgängerstreifen einen Task generiert.

Anwendung

Für eine Challenge benötigt es die Datei challange.json, welches die Challenge beschreibt und ein zweite Datei tasks.json, in dem sich die Tasks befinden.

Tasks generieren

```
# python TaskGenerator.py crosswalks.json
```

Challenge publizieren

```
# curl -u devuser:mylittlesony -vX
  POST http://maproulette.org/api/admin/challenge/
  crosswalk-detection/tasks -d @tasks.json
  --header "Content-Type: application:json"
```

Tasks publizieren

```
# curl -u devuser:mylittlesony -vX
  POST http://maproulette.org/api/admin/challenge/
  crosswalk-detection -d @challenge.json
  --header "Content-Type: application:json"
```

Als Hilfestellung zum Erstellen von MapRoulette Challanges gibt es ein empfehlenswertes Tutorial [vE15b].

4.2.4 Keras Training

Für das Training des Neuronalen Netzes steht ein eigenes Docker Image [Bü15] auf Docker Hub bereit. Das Image basiert auf einem offiziellen Nvidia Cuda Image und ist fähig mit einer Nvidia Grafikkarte zu arbeiten. Die Grafikkarte wird mit Hilfe des nvidia-docker Projekts geladen. CUDA muss dabei auf dem Host Rechner installiert sein.

Keras Image

Download des nvidia-docker Projekts

```
# git clone https://github.com/NVIDIA/nvidia-docker.git
# cd nvidia-docker
```

Herunterladen des Images

```
# docker pull sebu/keras_cuda
```

Start des Containers und mount der Grafikkarte Nummer 0

```
#GPU=0 ./nvidia-docker run -i -t sebu/keras_cuda /bin/bash
```

Achtung: Die Keras Bibliothek entwickelt sich ständig weiter. Auch die Interfaces der populärsten Klassen können sich ändern und haben sich auch schon während diesem Projekt verändert! In diesem Keras Docker Image ist der Stand von Keras installiert, mit dem wir gearbeitet haben. Keras bietet leider keine Versionierung an. Der von uns verwendete Stand ist auch auf der mitgelieferten CD erhältlich.

Ein Beispiel für das Training eines Neuronalen Netzes kann in examples/ConvnetTrainer.py und in den Keras eigenen Examples eingesehen werden.

Anhang A

Inhalt der CD

Der Inhalt der CD glieder sich folgendermassen:

```
CD
├── Studienarbeit.pdf
├── 0_AUFGABE
│   └── Original Aufgabenstellung
├── 1_CODE
│   └── GitHub Repository
├── 2_DOKUMENTATION
│   ├── Protokolle
│   └── Poster
├── 3_RESULTAT
│   └── Geodaten
└── 4_ANHANG
    └── Keras
```

Anhang B

Eigenständigkeitserklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt sind oder mit dem Betreuer schriftlich vereinbart wurden.
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Ort, Datum:

Rapperswil, 16. Dezember 2015

Namen, Unterschriften:

Severin Bühler

Samuel Kurath

Anhang C

Glossar

Bounding Box Eine Fläche, welche durch zwei Längengrade und zwei Breitengrade definiert ist. 26

CircleCI CircleCI ist ein Tool für Continuous Integration und Deployment. 56

Confusion Matrix Wahrheitsmatrix, Methode zur Bewertung von Klassifizieren. 18

Convnet Abkürzung für Convolutional Neural Network. 20

Convolutional Neural Network Neuronales Netz speziell für Bilderkennung. 20

Deep Learning Marketingbegriff für tiefe Neuronale Netze. 20

Docker Leichtgewichtige Virtualisierung. 33

Git Git ist ein Versionsverwaltungssystem. 60

GitHub GitHub ist ein webbasierter Dienst für Software-Entwicklungsprojekte. 60

Haar Feature-based Cascade Classifier Bilderklassifizierer aus OpenCV. 18

Inputbild RGB Bild mit der Grösse 50 x 50 Pixel, welches als Input eines Convnet dient. 51

Jira Jira ist eine webbasierte Anwendung für Projektmanagement. 60

Keras Python Bibliothek zum Training und der Verwendung von Neuronalen Netzen. 27

Mapquest Mapquest ist eine API für OpenStreetMap Daten. 46

MapRoulette Online Crowdsourcing-System zur spielerischen Überprüfung von Daten für OpenStreetMap. 22

Maptiler Maptiler bietet ein Python Skript zur Umrechnung von Koordinaten zu Tiles. 49

OpenCV C++ Bibliothek zur Verarbeitung von Bildern und Videos. 33

OpenStreetMap Online Karten Anwendung wie Google Maps. 25

Orthofotos Luftbilder, umgangssprachlich Satellitenbilder. 19

PyCharm PyCharm ist eine IDE von JetBrains für Python. 60

Quadkeys Quadkeys werden von Bing Maps für die Referenzierung zu ihren Tiles verwendet. 49

QuadTree Format zur Identifizierung von Tiles. 49

RQ RQ (Redis Queue) ist ist eine einfache Library für Python um Jobs in Redis einzureihen. 55

Scale-invariant Feature Transform Bildwiedererkenner aus OpenCV. 20

Tile Quadratisches Bild mit Karteninhalt, Kachel. 26

To-Fix Online Crowdsourcing-System zur spielerischen Überprüfung von Daten. 23

Anhang D

Literaturverzeichnis

[AlD15] AlDanial. Count Lines of Code. https://github.com/AlDanial/cloc, Aufgerufen Dezember 2015.

[Bü15] Severin Bühler. keras_cuda image for the nvidia-docker project. https://hub.docker.com/r/sebu/keras_cuda/, Aufgerufen Dezember 2015.

[Bal15] Thorsten Ball. TRAIN YOUR OWN OPENCV HAAR CLASSIFIER. http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html, Aufgerufen Dezember 2015.

[Car15] CartoDB. CartoDB is the Easiest Way to Map and Analyze Your Location Data. https://cartodb.com/, Aufgerufen Dezember 2015.

[Cho15] François Chollet. Keras. https://github.com/fchollet/keras, Aufgerufen Dezember 2015.

[Cir15] CircleCI. CircleCI. https://circleci.com/, Aufgerufen Dezember 2015.

[DRB15] Direktor der ETH-Bibliothek Dr. Rafael Ball. NEBIS. http://recherche.nebis.ch/, Aufgerufen Dezember 2015.

[Dri15] Vincent Driessen. RQ. http://python-rq.org/, Aufgerufen Dezember 2015.

[Fan15] Fantasma300580. Rational Unified Process. https://de.wikipedia.org/wiki/Rational_Unified_Process, Aufgerufen Dezember 2015.

[Goo15] Google. Google Scholar. http://scholar.google.ch/, Aufgerufen Dezember 2015.

[IEE15] IEEE. IEEXplore. http://ieeexplore.ieee.org/, Aufgerufen Dezember 2015.

[Inc15a] MapQuest Inc. Mapquest. http://www.mapquest.com/, Aufgerufen Dezember 2015.

[Inc15b] MapQuest Inc. Xapi Service Developer's Guide. http://open.mapquestapi.com/xapi/, Aufgerufen Dezember 2015.

[IS08] Yuichi Ishino and Hitoshi Saji. Extraction of road markings from aerial images. In *SICE Annual Conference, 2008*, pages 2180–2183. IEEE, 2008.

[Kar15] Andrej Karpathy. Convolutional Neural Networks. http://cs231n.github.io/convolutional-networks/, Aufgerufen Dezember 2015.

[KS15] Andrew Zisserman Karen Simonyan. Very Deep Convolutional Networks for Large-Scale Image Recognition. http://arxiv.org/abs/1409.1556, Aufgerufen Dezember 2015.

[Kur15] Samuel Kurath. Dockercontainer to run the workers of the OSM-Crosswalk-Detection project. https://hub.docker.com/r/murthy10/osm-crosswalk-detection/, Aufgerufen Dezember 2015.

[Lab15] OSM Lab. To-Fix. http://osmlab.github.io/to-fix/#/task/tigerdelta, Aufgerufen Dezember 2015.

[Mic15a] Microsoft. Bing Maps REST Services. https://msdn.microsoft.com/en-us/library/ff701713.aspx, Aufgerufen Dezember 2015.

[Mic15b] Microsoft. Bing Maps Tile System. https://msdn.microsoft.com/en-us/library/bb259689.aspx, Aufgerufen Dezember 2015.

[Př15] Klokan Petr Přidal. Tiles à la Google Maps: Coordinates, Tile Bounds and Projection. http://www.maptiler.org/google-maps-coordinates-tile-bounds-projection/, Aufgerufen Dezember 2015.

[San15] Salvatore Sanfilippo. Redis. http://redis.io/, Aufgerufen Dezember 2015.

[SE12] Turgay Senlet and Ahmed Elgammal. Segmentation of occluded sidewalks in satellite images. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 805–808. IEEE, 2012.

[vE15a] Martijn van Exel. MapRoulette. http://maproulette.org/, Aufgerufen Dezember 2015.

[vE15b] Martijn van Exel. MapRoulette Challenge Tutorial. https://gist.github.com/mvexel/b5ad1cb0c91ac245ea3f, Aufgerufen Dezember 2015.