



**HSR**

HOCHSCHULE FÜR TECHNIK  
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Bachelorarbeit, Abteilung Informatik

# strongMan

HSR Hochschule für Technik Rapperswil

Frühjahrssemester 2016

18. Juni 2016

*Autoren:* Bühler Severin & Kurath Samuel

*Betreuer:* Prof. Dr. Andreas Steffen

*Arbeitsperiode:* 22.02.2016 - 18.06.2016



## 0.1 Abstract

Die VPN Applikation ist weltweit stark verbreitet, nun soll es möglich sein die Konfiguration per graphischen Interface zu vereinfachen.

Weitere Informationen: <https://github.com/strongswan/>

## **0.2 Management Summary**

### **Ausgangslage**

Management per UI.

### **Vorgehen**

Erarbeitung der Lösung

### **Ergebnisse**

BA

### **Ausblick**

Erweitern

# Inhaltsverzeichnis

0.1	Abstract . . . . .	2
0.2	Management Summary . . . . .	3
0.3	Aufgabenstellung . . . . .	9
<b>1</b>	<b>Technischer Bericht</b>	<b>11</b>
1.1	Einführung . . . . .	12
1.1.1	Vision . . . . .	12
1.2	Stand der Technik . . . . .	13
1.2.1	Literaturrecherche . . . . .	13
1.3	Evaluation Usermanagement . . . . .	14
1.3.1	Auswertung . . . . .	14
1.4	Evaluation Zertifikatstypen . . . . .	15
1.4.1	Auswertung . . . . .	15
1.5	Evaluation Zertifikatsbibliothek . . . . .	16
1.5.1	Auswertung . . . . .	16
1.6	Umsetzungskonzept . . . . .	17
1.7	Resultate . . . . .	18
1.7.1	Zielerreichung . . . . .	18
1.7.2	Persönlicher Bericht . . . . .	19
1.7.3	Dank . . . . .	20
<b>2</b>	<b>Software Projektdokumentation</b>	<b>21</b>
2.1	Vision . . . . .	22
2.2	Anforderungsspezifikation . . . . .	23
2.2.1	Allgemeine Beschreibung . . . . .	23
2.2.2	Use Case . . . . .	23
2.2.3	Sequenzdiagramm . . . . .	27
2.2.4	Nichtfunktionale Anforderungen . . . . .	28
2.2.5	Analyse . . . . .	28
2.3	Implementation . . . . .	29
2.4	Test . . . . .	30
2.4.1	Unittests . . . . .	30
2.4.2	CircleCI . . . . .	30
<b>3</b>	<b>Projektmanagement</b>	<b>31</b>
3.1	Rollen und Verantwortlichkeiten . . . . .	32
3.1.1	Prof. Keller Stefan . . . . .	32
3.1.2	Bühler Severin . . . . .	32
3.1.3	Kurath Samuel . . . . .	33
3.2	Entwicklungsumgebung und Infrastruktur . . . . .	34

3.2.1	IDE (Integrated Development Environment)	34
3.2.2	SCM (Source Control Management)	34
3.2.3	CI (Continuous Integration)	34
3.2.4	Projektmanagement Tool	34
3.3	Planung	35
3.3.1	Phasen	35
3.3.2	Meilensteine	36
3.3.3	Zeitplanung	36
3.4	Risiken	38
3.4.1	Technische Risiken	38
3.4.2	Auswertung	39
3.5	Soll-Ist-Zeit-Vergleich	40
3.5.1	Inception	40
3.5.2	Elaboration1	41
3.5.3	Elaboration2	41
3.5.4	Construction1	41
3.5.5	Construction2	41
3.5.6	Transition	41
3.5.7	Übersicht	41
3.6	Codestatistik	42
3.6.1	Test Coverage	42
3.6.2	Codezeilen	42
3.7	Rational Unified Process (RUP)	43
<b>4</b>	<b>Softwaredokumentation</b>	<b>44</b>
4.1	Installation	45
4.1.1	Redis	45
4.1.2	Keras	45
4.1.3	Docker	46
4.2	Benutzerhandbuch	47
4.2.1	Suche der Fussgängerstreifen	47
4.2.2	Daten visualisieren	49
4.2.3	Challenge erstellen	50
4.2.4	Keras Training	51
<b>Anhang</b>		
<b>A</b>	<b>Inhalt der CD</b>	<b>52</b>
<b>B</b>	<b>Eigenständigkeitserklärung</b>	<b>53</b>
<b>C</b>	<b>Glossar</b>	<b>54</b>
<b>D</b>	<b>Literaturverzeichnis</b>	<b>56</b>

# Abbildungsverzeichnis

2.1	Use Case Diagramm . . . . .	24
3.1	Prof. Dr. Andreas Steffen . . . . .	32
3.2	Severin Bühler . . . . .	32
3.3	Samuel Kurath . . . . .	33
3.4	Gantt Chart . . . . .	37

# Tabellenverzeichnis

1.1	Resultate . . . . .	18
2.1	Aktoren und Stakeholder . . . . .	23
2.2	Eingabefelder VPN-Client . . . . .	26
2.3	Use Case Fully Dressed . . . . .	26
3.1	Risiken . . . . .	38
3.2	Risikoauswertung . . . . .	39
3.3	Phasen . . . . .	41
3.4	Test Coverage . . . . .	42
3.5	Codezeilen . . . . .	42



# Verzeichnis der Entscheide

1.1	Evaluation Usermanagement . . . . .	14
1.2	Evaluation Zertifikatstypen . . . . .	15
1.3	Evaluation Zertifikatsbibliothek . . . . .	16
3.4	PyCharm . . . . .	34
3.5	GitHub . . . . .	34
3.6	CircleCI . . . . .	34
3.7	Jira . . . . .	34
4.8	Docker . . . . .	46

## 0.3 Aufgabenstellung



### 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.

## 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** StrongSwan wird standardmässig per Konfigurationsdateien verwaltet, dies zielt stark auf eher versierte Nutzer und Administratoren ab.

**Vision** Es soll eine Applikation entstehen, mit der dieser komplexe Prozess erleichtert wird. Dabei wird auf ein graphisches Interface gesetzt.

## **1.2 Stand der Technik**

Android App.

### **1.2.1 Literaturrecherche**

#### **Suchquellen**

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

- a
- b
- c

#### **Auswertung**

-

#### **Fazit**

-

## **1.3 Evaluation Usermanagement**

### **1.3.1 Auswertung**

-

#### **Entscheid 1. Evaluation Usermanagement**

-

## **1.4 Evaluation Zertifikatstypen**

### **1.4.1 Auswertung**

-

### **Entscheid 2. Evaluation Zertifikatstypen**

-



## **1.5 Evaluation Zertifikatsbibliothek**

### **1.5.1 Auswertung**

-

#### **Entscheid 3. Evaluation Zertifikatsbibliothek**

-

## **1.6 Umsetzungskonzept**

Die Grundidee für die Umsetzung des Management Tools:

1. 1
2. 2

## 1.7 Resultate

### 1.7.1 Zielerreichung

-

#### 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 ?? auf der Seite ??.
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 Bekanntheit bei der Community durch. Mehr dazu unter dem Abschnitt ?? auf der Seite ??.
Erstellung einer Challenge für das Crowdsourcing-System anhand der gesammelten Daten.	Eine Challenge wurde für MapRoulette generiert und publiziert.

Tabelle 1.1: Resultate

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

## **1.7.2 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
- Latex

### **1.7.3 Dank**

-

## **Kapitel 2**

# **Software Projektdokumentation**

## **2.1 Vision**

Die Vision ist unter dem Abschnitt 1.1.1 auf der Seite 12 zu finden.

## 2.2 Anforderungsspezifikation

### 2.2.1 Allgemeine Beschreibung

Im generellen sind zwei Anwendungsszenarien denkbar:

- VPN-Client
- VPN-Gateway

Dabei ist der VPN-Client eine **Muss**-Anforderung und der VPN-Gateway eine **Kann**-Anforderung.

#### VPN-Client

Die Applikation wird von einem Standard-Nutzer verwendet. Dieser soll VPN Tunnels zu Gateways konfigurieren können und die Tunnels starten und stoppen. Die Konfigurationsmöglichkeiten sind beschränkt, als Richtwert wird der strongSwan Android Client verwendet.

#### VPN-Gateway

Der Gateway ist auf Systemadministratoren ausgerichtet. Es soll möglich sein per grafischem Interface strongSwan zu konfigurieren und Tunnels einzurichten, welche als Gateway genutzt werden.

### 2.2.2 Use Case

#### Aktoren und Stakeholder

Aktor	Tätigkeit
User	<ul style="list-style-type: none"><li>• Konfiguriert VPN-Tunnel als Client</li><li>• Startet und stoppt VPN-Tunnel</li></ul>
Administrator	<ul style="list-style-type: none"><li>• Konfiguriert VPN-Tunnel als Gateway</li><li>• Startet und stoppt VPN-Tunnel</li></ul>

Tabelle 2.1: Aktoren und Stakeholder



## Use Case Diagramm

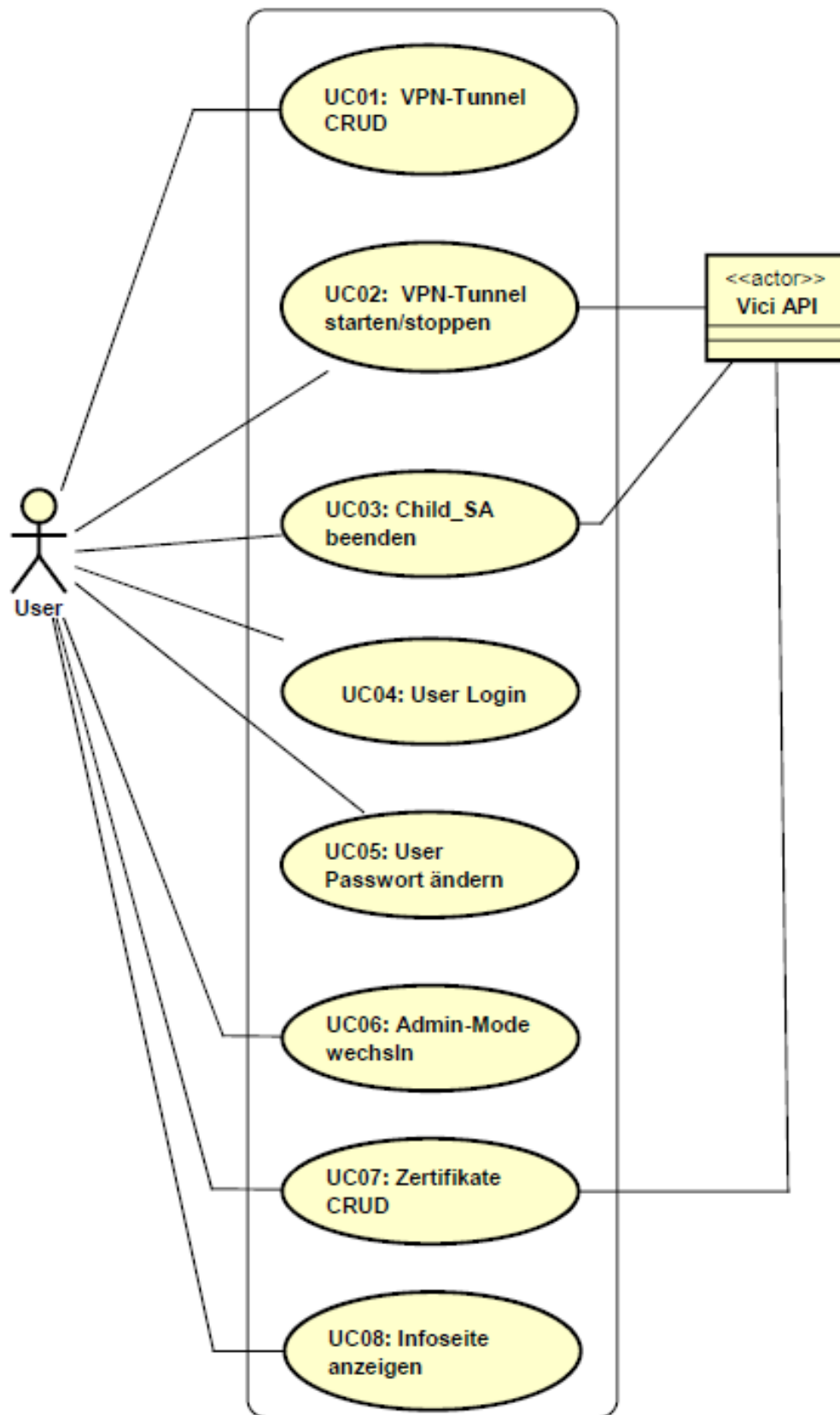


Abbildung 2.1: Use Case Diagramm

## Use Cases Brief

Alle hier definierten Use Cases haben auch ein entsprechendes Mockup im Anhang.

### UC01: VPN-Tunnel CRUD

Der User kann einen VPN-Tunnel erfassen / konfigurieren. Dabei hat er eine Auswahl von verschiedenen vordefinierten Tunneltypen. Jeder Tunneltyp hat eigene Konfigurationsfelder, die der User ausfüllen muss. Die Tunnel-Übersichtsseite stellt die Hauptseite der Applikation dar. Dort können die Tunnels bearbeitet und gelöscht werden.

### UC02: VPN-Tunnel starten/stoppen

Der User kann einen erfassten VPN-Tunnel starten und stoppen. Dabei wird die Konfiguration über die Vici API geladen. Falls ein VPN-Tunnel nicht aufgebaut werden kann, soll eine passende Fehlermeldung angezeigt werden.

### UC03: Child\_SA beenden

Jeder VPN-Tunnel kann mehrere Child\_SA enthalten. Dieser werden in der Hauptseite angezeigt und können vom User beendet werden. Dieser Use Case interagiert mit der VICI Schnittstelle.

### UC04: User Login

Der User loggt sich zu Beginn beim Webseiten Aufruf mit einem Passwort ein. Es existiert dabei nur ein User mit Passwort.

### UC05: User Passwort ändern

Sobald der User eingeloggt ist, hat er die Möglichkeit, sein Passwort zu ändern. Dabei gibt er sein altes Passwort einmal und sein neues Passwort zweimal ein.

### UC06: Admin-Mode wechseln

Das Userinterface unterscheidet zwischen zwei Modis: User- & Admin-Mode. Der Mode kann durch einem Klick auf einen Button gewechselt werden. Der Admin-Mode stellt einige Gateway-spezifische Funktionalitäten zusätzlich zur Verfügung, welche der User zur einfacheren Bedienung nicht sieht.

### UC07: Zertifikate CRUD

Dem User wird eine Zertifikatsverwaltung zur Verfügung gestellt. Er kann Zertifikate und Private Key's in den gängigen Formaten uploaden, anschauen, updaten (Passwort ändern) und wieder löschen. Die Dateien können mit einem Passwort verschlüsselt sein. Dieser Use Case interagiert unter Umständen mit der VICI Schnittstelle.

### UC08: Infoseite anzeigen

Die Infoseite zeigt dem eingeloggten User verschieden Informationen über das installierte System wie Charon Version, installierte Plugins usw.

## Definition Konfigurationsmöglichkeiten UC01

### VPN-Client

Folgende Verbindungsarten müssen unterstützt werden:

- IKEv2 Zertifikat
- IKEv2 EAP (Benutzername/Passwort)
- IKEv2 Zertifikat + EAP (Benutzername/Passwort)
- IKEv2 EAP-TLS (Zertifikat)

Name	swanctl	vici
Profilname	connections.<conn>	<IKE_SA config name>
Typ	"Verbindungsarten"	"Verbindungsarten"
Gateway	connections.<conn>.remote_addrs	remote-host
EAP Username	connections.<conn>.local<suffix>.eap_id (Ref: secrets.eap<suffix>.id<suffix>)	<IKE>.remote-eap-id
EAP Passwort	secrets.eap<suffix>.secret	<secret>.data
Gateway-Port	connections.<conn>.remote_port	<IKE>.remote-port
CA-Zertifikat	connections.<conn>.remote<suffix>.cacerts	
User-Zertifikat	connections.<conn>.local<suffix>.certs	

Tabelle 2.2: Eingabefelder VPN-Client

## Eingabefelder

### VPN-Gateway

### **2.2.3 Sequenzdiagramm**

**User**

Das Sequenzdiagramm beschreibt...

## **2.2.4 Nichtfunktionale Anforderungen**

**Funktionalität**

**Sicherheit**

**Interoperabilität**

**Richtigkeit**

**Zuverlässigkeit**

**Wiederherstellbarkeit**

**Fehlertoleranz**

**Availability**

**Benutzbarkeit**

**Robustheit**

**Effizienz**

**Supportability**

**Internationalization**

## **2.2.5 Analyse**

**Beschreibung Domain Modell**

-

## 2.3 Implementation

-

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

## 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. Dr. Andreas Steffen  
Professor, Institutsleiter ITA

### **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 4. 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 5. 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 6. 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 7. 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

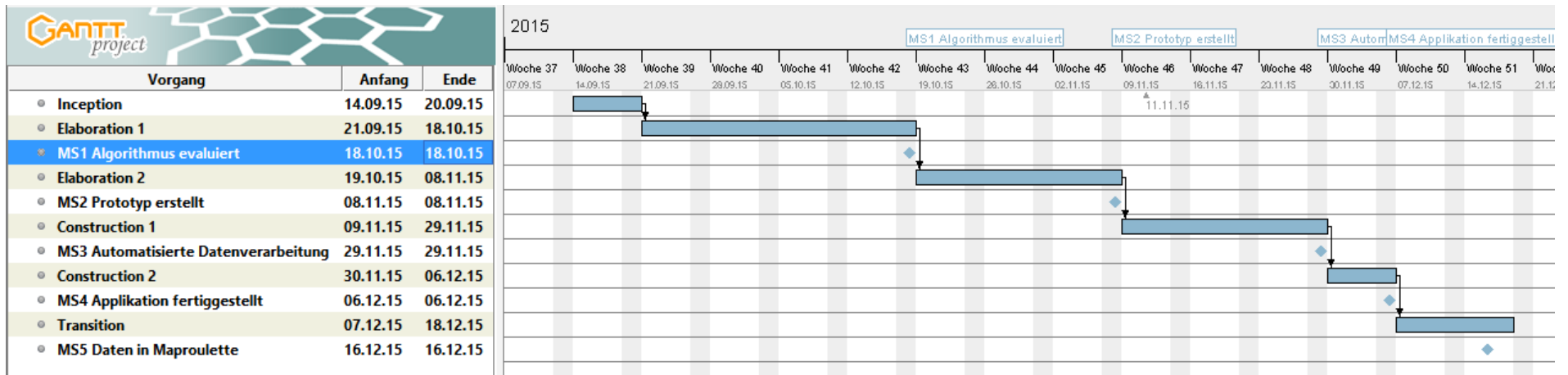


Abbildung 3.4: Gantt Chart

## 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	maximale 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 verschiedenen 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

### 3.5.2 Elaboration1

Start: 23.09.2015 Ende: 19.10.2015

### 3.5.3 Elaboration2

Start: 19.10.2015 Ende: 04.11.2015

### 3.5.4 Construction1

Start: 04.11.2015 Ende: 25.11.2015

### 3.5.5 Construction2

Start: 04.11.2015 Ende: 11.11.2015

### 3.5.6 Transition

Start: 11.12.2015 Ende: 18.12.2015

### 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)

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 agileres 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.

#### Installation

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

#### Konfiguration

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

#### Starten

```
1 # 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

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

#### Installation Keras

```
1 # apt-get install -y libhdf5-dev python-h5py python-yaml
2 # pip install --upgrade six
3 # cd /root
4 # git clone https://github.com/fchollet/keras.git
5 # cd keras
6 # 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 andererseits auch nicht wirklich trivial sind..

#### Entscheid 8. Docker

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

#### 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

```
1 # sudo apt-key adv --keyserver hkp://p80.pool.sks
2 --keyservers.net:80 --recv-
3 keys 58118E89F3A912897C070ADB76221572C52609D
4 # echo 'deb https://apt.dockerproject.org/repo ubuntu-precise main'
5 >> /etc/apt/sources.list.d/docker.list
6 # apt-get update
7 # apt-cache policy docker-engine
8 # sudo apt-get install linux-image-extra-$(uname -r)
```

##### Docker installieren

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

#### Starten

```
1 # sudo service docker start
2 # 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 45
- Docker: Abschnitt 4.1.3 auf der Seite 46

#### 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 ?? auf der Seite ??



## Anwendung

Dank Docker kann unsere Applikation innert Minuten gestartet werden.

### Docker Pull

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

### Manager

```
1 # docker run murthy10/osm-crosswalk-detection REDIS_IP_ADDR
2 --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

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

Jobworker können auf beliebig vielen Rechnern gestartet werden.

### Resultworker

```
1 # docker run murthy10/osm-crosswalk-detection REDIS_IP_ADDR
2 --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 `challenge.json`, welches die Challenge beschreibt und eine zweite Datei `tasks.json`, in dem sich die Tasks befinden.

#### Tasks generieren

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

#### Challenge publizieren

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

#### Tasks publizieren

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

Als Hilfestellung zum Erstellen von MapRoulette Challenges 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

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

#### Herunterladen des Images

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

#### Start des Containers und mount der Grafikkarte Nummer 0

```
1 #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, 14. März 2016

Namen, Unterschriften:

Severin Bühler

Samuel Kurath

# Anhang C

## Glossar

**Bounding Box** Eine Fläche, welche durch zwei Längengrade und zwei Breitengrade definiert ist. 23

**CircleCI** CircleCI ist ein Tool für Continuous Integration und Deployment. 54

**Confusion Matrix** Wahrheitsmatrix, Methode zur Bewertung von Klassifizieren. 15

**Convnet** Abkürzung für Convolutional Neural Network. 17

**Convolutional Neural Network** Neuronales Netz speziell für Bilderkennung. 17

**Deep Learning** Marketingbegriff für tiefe Neuronale Netze. 17

**Docker** Leichtgewichtige Virtualisierung. 30

**Git** Git ist ein Versionsverwaltungssystem. 58

**GitHub** GitHub ist ein webbasierter Dienst für Software-Entwicklungsprojekte. 58

**Haar Feature-based Cascade Classifier** Bilderklassifizierer aus OpenCV. 15

**Inputbild** RGB Bild mit der Grösse 50 x 50 Pixel, welches als Input eines Convnet dient. 48

**Jira** Jira ist eine webbasierte Anwendung für Projektmanagement. 58

**Keras** Python Bibliothek zum Training und der Verwendung von Neuronalen Netzen. 24  
. 43

**MapRoulette** Online Crowdsourcing-System zur spielerischen Überprüfung von Daten für Open-StreetMap. 19

**Maptiler** Maptiler bietet ein Python Skript zur Umrechnung von Koordinaten zu Tiles. 45

**OpenCV** C++ Bibliothek zur Verarbeitung von Bildern und Videos. 30

**OpenStreetMap** Online Karten Anwendung wie Google Maps. 22

**Orthofotos** Luftbilder, umgangssprachlich Satellitenbilder. 16

**PyCharm** PyCharm ist eine IDE von JetBrains für Python. 58

**Quadkeys** Quadkeys werden von Bing Maps für die Referenzierung zu ihren Tiles verwendet.  
46

**QuadTree** Format zur Identifizierung von Tiles. 46

**RQ** RQ (Redis Queue) ist ist eine einfache Library für Python um Jobs in Redis einzureihen. 53

**Scale-invariant Feature Transform** Bildwiedererkenner aus OpenCV. 17

**Tile** Quadratisches Bild mit Karteninhalt, Kachel. 23

**To-Fix** Online Crowdsourcing-System zur spielerischen Überprüfung von Daten. 20



# Anhang D

## Literaturverzeichnis

- [AlD15] AlDanial. Count Lines of Code. <https://github.com/AlDanial/cloc>, 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.
- [Bü15] Severin Bühler. keras\_cuda image for the nvidia-docker project. [https://hub.docker.com/r/sebu/keras\\_cuda/](https://hub.docker.com/r/sebu/keras_cuda/), 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](https://de.wikipedia.org/wiki/Rational_Unified_Process), Aufgerufen Dezember 2015.
- [Goo15] Google. Google Scholar. <http://scholar.google.ch/>, Aufgerufen Dezember 2015.
- [IEE15] IEEE. IEEExplore. <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] Klokán 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.