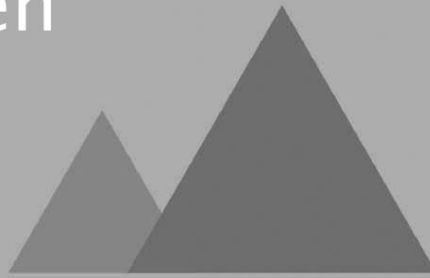


Bitte Platzhalter löschen  
und durch eigenes Bild  
ersetzen



## Detektion von liegendem Schnee mit Bildanalyse

### Bachelorthesis

Studiengang:	BSc Informatik (CPVR)
Autor:	Marko Public
Betreuer:	Marcus Hudritsch
Auftraggeber:	Lorenz Martin - Daniel Bättig - ASTRA - MeteoSchweiz
Experte:	Mathis Marugg
Datum:	19. Januar 2017

# Management Summary

## Ausgangslage

Das Bundesamt für Strassen (ASTRA) entwickelt in Zusammenarbeit mit der BFH ein neues Frühwarnsystem für Strassenglätte, das 2018 in Betrieb gehen soll. Aus gemessenen Meteo-Daten kann für die meisten Situationen bereits jetzt die Wahrscheinlichkeit für Glätte errechnet werden.

Es gibt aber ein Szenario, das aus Wetterdaten nicht berechenbar ist: Wenn Schnee neben der Fahrbahn liegt, in der Nachmittagssonne schmilzt und auf den Asphalt fliesst. Gegen Abend kann die Temperatur wieder unter den Gefrierpunkt sinken und das Schmelzwasser auf der Fahrbahn zum Gefrieren bringen.

Die Idee ist nun dieses Szenario frühzeitig zu erkennen: Mit Methoden der Bildverarbeitung und der Merkmalsextraktion sowie Webcam-Bildern soll automatisch bestimmt werden, ob neben der Strasse Schnee liegt oder nicht.

## Datenaufbereitung

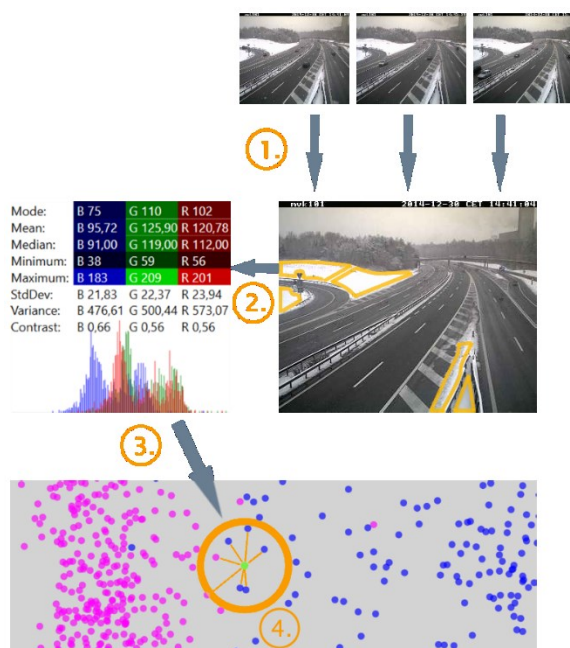
Für den Aufbau einer Wissensbasis wurden über mehrere Winter Webcam-Bilder gesammelt. Die Bilder wurden zu Kategorien (Schnee / kein Schnee) zugeordnet und auf Bildausschnitte reduziert. Aus den Bildausschnitten konnten Merkmale wie Histogramm, Durchschnittsfarbe pro Farbkanal und Kontrast extrahiert und in einer Datenbank gespeichert werden.

Diese Datenbasis stellt die Grundlage, um Referenzwerte für verschiedene Tageszeiten, Sonnenstände und Wetterlagen (Sonnig / Niederschlag / Nebel) zu gewinnen – jeweils für beide Kategorien.

Der Ansatz ist nun, dass ein Distanzmass Anwendung findet, das aussagt, ob ein Bild näher an den Referenzwerten der einen oder der anderen Kategorie liegt. Dieses Distanzmass stützt sich auf die extrahierten statistischen Merkmale aus der Datenbasis.

## Algorithmus

1. In kurzer Reihenfolge werden Bilder von der Webcam heruntergeladen und kombiniert. So ist sichergestellt, dass keine Autos auf dem Bild sind, welche die Farbe des betrachteten Bildausschnitts verfälschen könnten
2. Es folgt die statistische Auswertung aller Bildausschnitte
3. Schliesslich die Suche nach den nächsten Nachbarn unter allen Referenzwerten für jeden betrachteten Bildausschnitt
4. Das Bild wird derjenigen Kategorie zugewiesen, zu welcher eine Mehrzahl der nächsten Nachbarn angehört



## Ergebnisse

Im Rahmen der Arbeit entstand ein mächtiges Werkzeug als .NET-Applikation, das Aufbau und Pflege der Datenbasis unterstützt und das den parametrierbaren Algorithmus systematisch testet. Der mehrmals optimierte Algorithmus erreicht eine genügend hohe Erkennungsrate für den Einsatz in einer produktiven Umgebung.

# Inhaltsverzeichnis

1	Einleitung	5
2	Kameras	6
2.1	Aufzeichnung der Bilder	6
2.2	Standorte	6
2.2.1	Karte	10
2.2.2	Zeitdiskrepanz zwischen Dateinamen und Meta-Daten	10
2.3	Beschaffenheit der JPEG-Dateien	11
3	Projekt 2	12
3.1	Technologie	12
3.2	Metadaten in Datenbank	12
3.3	Segmentierung des Bildes	13
3.3.1	Polygon Editor	13
3.4	Bereinigung von Autos	14
3.4.1	Algorithmus	14
3.5	Naive Schneedetektion	18
3.6	Resultate	18
3.7	Weitere Themen	19
3.7.1	Wetterdaten	19
3.7.2	Intelligenter Cron-Job	19
4	Bachelorarbeit	20
4.1	Verbesserung zu Projekt 2	20
4.2	F-Test	20
4.3	Stichprobe aufbauen	20
4.3.1	Kategorien	21
4.3.2	Programm	22
4.3.3	Anpassungen Galerie Modul	23
4.3.4	Erfassung der Kategorien	23
4.3.5	Verhältnis	23
4.3.6	Wahrscheinlichkeit für eingeschränkte Sichtverhältnisse	24
4.4	Patches pro Bild	<b>Error! Bookmark not defined.</b>
4.4.1	Polygon Editor	<b>Error! Bookmark not defined.</b>
4.4.2	Neben der Strasse	<b>Error! Bookmark not defined.</b>
4.4.3	Sonneneinstrahlungs-Patch	<b>Error! Bookmark not defined.</b>
4.4.4	Statistische Werte pro Patch	26
4.4.4.1	Histogram	<b>Error! Bookmark not defined.</b>
4.4.4.2	Mode	27
4.4.4.3	Mean	27
4.4.4.4	Median	27
4.4.4.5	Minimum	27
4.4.4.6	Maximum	27
4.4.4.7	Standard Abweichung	27
4.4.4.8	Varianz	27
4.4.4.9	Kontrast	27
4.4.4.10	Werte ohne Hilfe von openCV berechnet	27
4.4.5	Patch Crop ohne schwarzen Rand	27
4.5	Logit Regression mit Daniel Bättig	<b>Error! Bookmark not defined.</b>
4.6	Albedo-Wert	<b>Error! Bookmark not defined.</b>
4.7	OneDrive Probleme	28
4.7.1	Anderer Computer	28
4.7.2	Zugriff	28
4.7.3	Backups	28
4.7.3.1	Azure	28

4.8 Kombinieren	27
4.8.1 2h Slots pro Woche kombinieren und abspeichern	27
4.8.2 Mean	27
4.8.3 Median	27
4.9 Plots	27
4.9.1 Clickable	27
4.9.2 Combined Plots	27
4.10 Klassifizierer	27
4.10.1 NNN	27
4.10.2 Optimierungen	27
4.10.2.1 Input Kombinieren	27
4.10.2.2 Abstand ausrechnen aus 3 Inputs und entfernten verwerfen	27
4.10.3 F-Werte	27
4.10.4 Ergebnisse	28
4.10.5 Echte Verhältnisse	28
5 Schlussfolgerungen/Fazit	29
6 Abbildungsverzeichnis	30
7 Tabellenverzeichnis	30
8 Glossar	30
9 Literaturverzeichnis	30
10 Anhang	31
11 Selbständigkeitserklärung	32

# 1 Einleitung

Im Auftrag der ASTRA betreibt das Institut für Risiko- und Extremwertanalyse der Berner Fachhochschule ein Strassenglätte-Prognosen-System (SGPS). Dieses bezieht Wetterdaten von SwissMeteo und modelliert Prognosen. Das Ziel ist rechtzeitig und korrekt zu warnen, wenn sich der Strassenzustand sich verschlechtert und Schneepflüge und Salzstreuer rechtzeitig ausrücken können.

Aus den Wetterdaten lässt sich genau und örtlich granular voraussagen, ob in den nächsten Stunden Schnee geräumt werden muss. Es gibt eine Gegebenheit, die das Ausrücken des Winterdiensts unverzichtbar macht aber aus Wetterdaten nicht errechnet werden kann: Neben der Strasse, zum Beispiel auf dem Pannenstreifen oder auf benachbarten Oberflächen, liegt Schnee. Dieser schmilzt tagsüber bei Temperaturen über null fließt auf die Fahrbahn. Nach Sonnenuntergang und tieferen Temperaturen, kann dieses Wasser zufrieren und Strassenglätte verursachen. Um bei dieser Situation zuverlässig Alarm zu geben sollen die Webcams, die den Verkehrsfluss überwachen, genutzt werden.

- ⇒ P\_Schnee: 1 wenn Schneehöhe > 2cm / 0 wenn Schneehöhe < 2cm
- ⇒ Kritische Zeit: Stosszeiten
- ⇒ Das Resultat dieser Projektarbeit zeigt, dass die Bilder der Kameras durchaus geeignet sind, um diese Informationslücke zu füllen.

## 2 Kameras

Für die Überwachung des Verkehrsflusses betreibt das Bundesamt für Strassen (ASTRA) Webcams zur Überwachung des Verkehrsflusses. Die Kameras sind an verschiedenen Orten im Nationalstrassennetz platziert. Die Geräte sind an Kandelabern, Brücken und Verkehrsschildern montiert und sind rund um die Uhr in Betrieb.

### 2.1 Aufzeichnung der Bilder

Die Aufzeichnung der Bilder begann anfangs Dezember 2014. Ein Cron-Job auf einem BFH-Server namens 'athena' stösst ein Skript an, dass alle 10 Minuten das aktuelle Bild der Webcam als JPG-Datei herunterlädt und das Bild auf einem File-Share ablegt. So funktioniert das Bash-Skript:

```
#!/bin/bash

# URL der Online-Webcams der ASTRA
url=http://www.astramobcam.ch/kamera
# Ziel: File-Share auf athena
dest=/srv/athena.bfh.ch/projects/astra/

# Liste der Kameras erstellen. Ziel-Ordner müssen existieren!
cams=(mvk021 mvk101 mvk105 mvk107 mvk110 mvk120 mvk122 mvk131)

# Aktuelles Datum auslesen in der Form JahrMonatTag_StundeMinuteSekunde
date=$(date -u +%Y%m%d_%H%M%S)
echo $date

# Zugriff auf File-Share prüfen
# Wenn nicht erreichbar, wird abgebrochen und eine Meldung per Mail versendet
if [ ! -d ${dest} ]; then
    echo ${date} | mail -s "astrafetch - storage destination not available" vep2@bfh.ch --
    -f vep2@bfh.ch
    exit -1
fi

# Pro Kamera in der Liste
for cam in ${cams[@]}; do
    # Leere Log-Datei erstellen
    log=${dest}/${cam}/${cam}_${date}.log
    # Leere JPG-Datei erstellen
    img=${dest}/${cam}/${cam}_${date}.jpg
    # Kamerabild per WGET herunterladen.
    # Daten in JPG-Datei schreiben und Log in Log-Datei schreiben
    /usr/bin/wget ${url}/${cam}/live.jpg -O ${img} -o ${log}
done
```

Über zwei Winter wurden für die ausgewählten 8 Kameras insgesamt 418'111 Bilder gesammelt. Da nur die Kamerabilder im Winter von Interesse sind, lief der Cron-Job von Dezember 2014 bis Mai 2015 und wider vom September 2015 bis April 2016.

### 2.2 Standorte

Die ursprünglich gewählten Standorte befinden sich im Grossraum Bern und entlang der Achse Thunersee-Brienzersee-Brünigpass. In Bern zeigen die Kameras auf die Autobahn A1 und A6, während die andere Gruppe auf der Nationalstrasse A8 verteilt ist. Ursprünglich ging man davon aus, dass die Kameras an einem Ort fest installiert sind.

Bei der Verarbeitung der Bilder fiel aber auf, dass gewisse Kameras zwar die Bezeichnung behielten aber während der Datensammlung physisch verschoben wurden. Die Bezeichnungen der Kameras lassen darauf schliessen, dass Geräte in unregelmässigen Abständen bewegt werden. Das Kürzel 'mvk' bedeutet Mobile-Video-Kamera. Dieser Umstand war bei Projektbeginn nicht bekannt.

Vier der acht Kameras lieferten im Verlauf der Datenaufnahme somit Bilder aus verschiedenen Perspektiven. So kamen Autobahnabschnitte in Kirchberg (A1) und bei Grenchen (A5) hinzu. Somit existieren Daten für folgende Kameras:



#### **mvk021**

Bezeichnung: A1 Grauholz > Bern

Koordinaten: 46°59'20.1"N 7°28'24.0"E

Aufzeichnung Winter 14/15: 2014-12-02 - 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 - 2015-12-04

Nach dem 4. Dezember 2015 lieferte die Kamera unter der gleichen Bezeichnung Bilder von einem anderen Ort aus. Für die Weiterverarbeitung wird die Bezeichnung mvk022 verwendet.



#### **mvk022**

Bezeichnung: A5 Pieterlen > Biel

Koordinaten: 47°10'06.2"N 7°21'33.3"E

Aufzeichnung Winter 15/16: 2015-12-07 - 2016-04-15



#### **mvk101**

Bezeichnung: A1 Weyermannshaus > Grauholz

Koordinaten: 46°57'02.9"N 7°24'29.2"E

Aufzeichnung Winter 14/15: 2014-12-02 - 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 - 2015-11-12

Nach dem 11. November 2015 lieferte die Kamera unter der gleichen Bezeichnung Bilder von einem anderen Ort aus. Für die Weiterverarbeitung wird die Bezeichnung mvk102 verwendet.



#### **mvk102**

Bezeichnung: A1 Tunnel Kirchb. Port. Zürich

Koordinaten: 47°05'12.4"N 7°34'25.9"E

Aufzeichnung Winter 15/16: 2015-11-16 - 2016-04-15





#### **mvk105**

Bezeichnung: A6 Wankdorf > Thun

Koordinaten: 46°57'58.2"N 7°28'15.0"E

Aufzeichnung Winter 14/15: 2014-12-02 – 2015-06-10

Aufzeichnung Winter 15/16: 2015-10-20 – 2015-10-28

Nach dem 28. Oktober 2015 lieferte die Kamera unter der gleichen Bezeichnung Bilder von einem anderen Ort aus. Für die Weiterverarbeitung wird die Bezeichnung mvk106 verwendet.



#### **mvk106**

Bezeichnung: A1 Rampendosierung Kirchberg (Ost)

Koordinaten: 47°04'45.6"N 7°34'20.5"E

Aufzeichnung Winter 15/16: 2015-11-04 – 2016-04-15



#### **mvk107**

Bezeichnung: A6 Ostring > Wankdorf

Koordinaten: 46°57'04.7"N 7°28'19.5"E

Aufzeichnung Winter 14/15: 2014-12-02 – 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 – 2015-10-28

Nach dem 28. Oktober 2015 lieferte die Kamera unter der gleichen Bezeichnung Bilder von einem anderen Ort aus. Für die Weiterverarbeitung wird die Bezeichnung mvk108 verwendet.



#### **mvk108**

Bezeichnung: A1 Rampendosierung Kirchberg (West)

Koordinaten: 47°04'45.6"N 7°34'20.5"E

Aufzeichnung Winter 15/16: 2015-11-04 – 2016-04-15





#### **mvk110**

Bezeichnung: A8 Gnoll (Brünig) > Luzern

Koordinaten: 46°45'05.5"N 8°07'42.0"E

Aufzeichnung Winter 14/15: 2014-12-02 – 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 – 2016-04-15



#### **mvk120**

Bezeichnung: A8 Brienz > LU

Koordinaten: 46°44'33.3"N 8°03'46.0"E

Aufzeichnung Winter 14/15: 2014-12-02 – 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 – 2016-04-15



#### **mvk122**

Bezeichnung: A8 Faulensee > LU

Koordinaten: 46°40'04.6"N 7°43'20.5"E

Aufzeichnung Winter 14/15: 2014-12-18 – 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 – 2016-04-12



#### **mvk131**

Bezeichnung: A8 Soliwalddtunnel Süd > LU

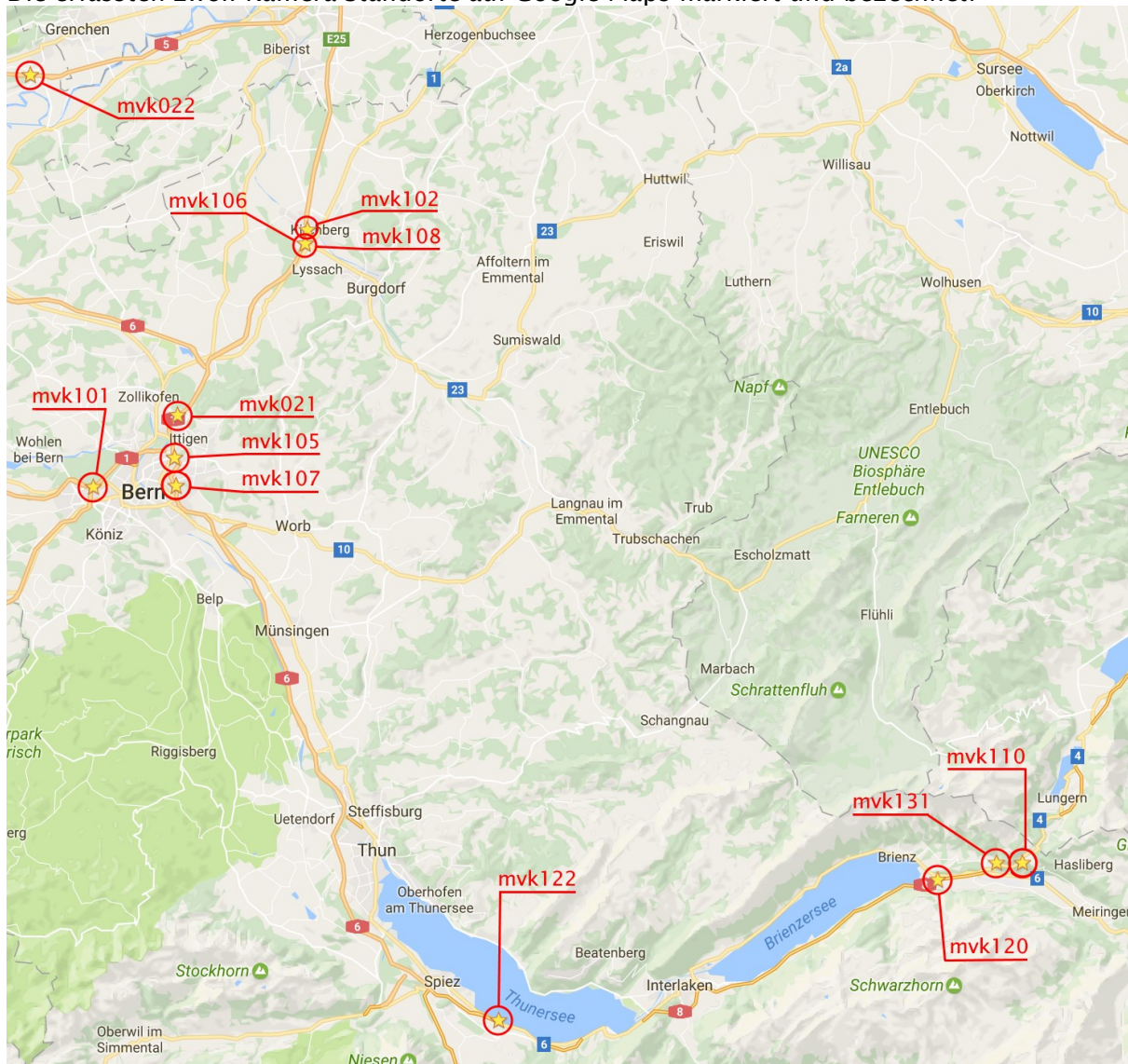
Koordinaten: 46°45'05.5"N 8°06'29.8"E

Aufzeichnung Winter 14/15: 2014-12-18 – 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 – 2016-04-15

## 2.2.1 Karte

Die erfassten zwölf Kamera-Standorte auf Google Maps markiert und bezeichnet:



## 2.2.2 Zeitdiskrepanz zwischen Dateinamen und Meta-Daten

Das Skript, welches die Bilder herunterlädt, gab jedem Bild einen Dateinamen. Dieser ist eine Kombination aus Kameraname und Uhrzeit des Downloads. Bei näherer Betrachtung der Bilder entstand der Eindruck, dass die Zeit im Dateinamen um eine oder zwei Stunden von der echten Aufnahmezeit abweicht. Diese Diskrepanz konnte aus Analyse des Sonnenstandes (Sonnenaufgang und Sonnenuntergang) auf den Bildern rekonstruiert werden. Beim Abgleich des Sonnenstandes mit der echten Uhrzeit kam die Web-Applikation von [timeanddate.com](http://timeanddate.com) zu Hilfe.

Die Uhrzeit im Dateinamen entspricht der Systemzeit des Servers, welche nicht mit der 'echten' Aufnahmezeit übereinstimmt. Glücklicherweise speichert die Kamera, neben den eigentlichen Bilddaten auch weiter Metainformationen in die Datei. Die JPG-lässt sich in einem Texteditor öffnen und gibt viele zusätzliche Information in einem Header-Teil der Datei preis. Unter den Metainformationen befindet sich auch eine Sektion namens 'Fingerprint', die den genauen Aufnahmezeitpunkt beinhaltet.

Auszug aus dem Header-Teil	Erklärung der relevanten Schlüsselwörter
[...] SECTION FINGERPRINT VER=1.0	Version: Version der Kamera / Firmware

PRD=MOBOTIX FRM=381865 DAT=2014-12-30 TIM=14:41:04.009 TZN=CET TIT=1419946864.009 ENO=0 IMT=IMAGE ENDSECTION FINGERPRINT [...]	Producer: Hersteller der Kamera / Firmware  Datum Time: Zeit Timezone: Zeitzone Timestamp: Unix-Zeitstempel
---	--

Beim Registrieren der Bildinformationen in Datenbank wurde nicht der Dateiname, sondern die korrekten Informationen aus dem Header-Teil berücksichtigt.

## 2.3 Beschaffenheit der JPEG-Dateien

Die gesammelten Bilddateien sind alles JPG-Dateien. Sie sind 352 Pixel breit und 288 Pixel hoch, haben eine Auflösung von 96dpi und eine Farbtiefe von 24bit (True Color). Pro Farbkanal (Rot, Grün, Blau) und Bildpunkt in einem Byte (0-255) kodiert. Die Bilddateien beinhalten, abgesehen vom proprietären Header-Teil, keine weiteren Metadaten in der Form von Exif-Informationen.

Von blossen Auge wirken die Bilder stark komprimiert und weisen sichtbare Kompressionsartefakte (Blockartefakte, Unschärfe) auf. Abhängig vom Bildinhalt sind die Dateien zwischen 6 und 25 Kilobyte gross. Die kleinen Dateien entstehen hauptsächlich nachts, wenn die Webcams von einem Farbbild auf Schwarz-Weiss umschalten.

## 3 Projekt 2

Im Projekt 2, mit Durchführung im Herbstsemester 2015, war der Ansatz die Schneedetektion ohne Vorwissen über Lichtverhältnisse auf dem Bild durchzuführen. Folgende Themen standen während der Projektarbeit im Zentrum:

- Metadaten des Bildarchivs in eine Datenbank abspeichern
- Autos bereinigen von Aufnahmen
- Manuelle Segmentierung des Bildes
- Detektion mit globalen Schwellwerten pro Bild und Segment

### 3.1 Technologie

Während dem Projekt ist eine Applikation entstanden, die die grosse Datenmenge übersichtlicher macht, die Segmentierung der Bilder ermöglicht und die Schneedetektion für verschiedenen Szenarien testweise ausführbar macht. Bei der Entwicklung kamen folgende Technologien und Plattformen zum Einsatz:

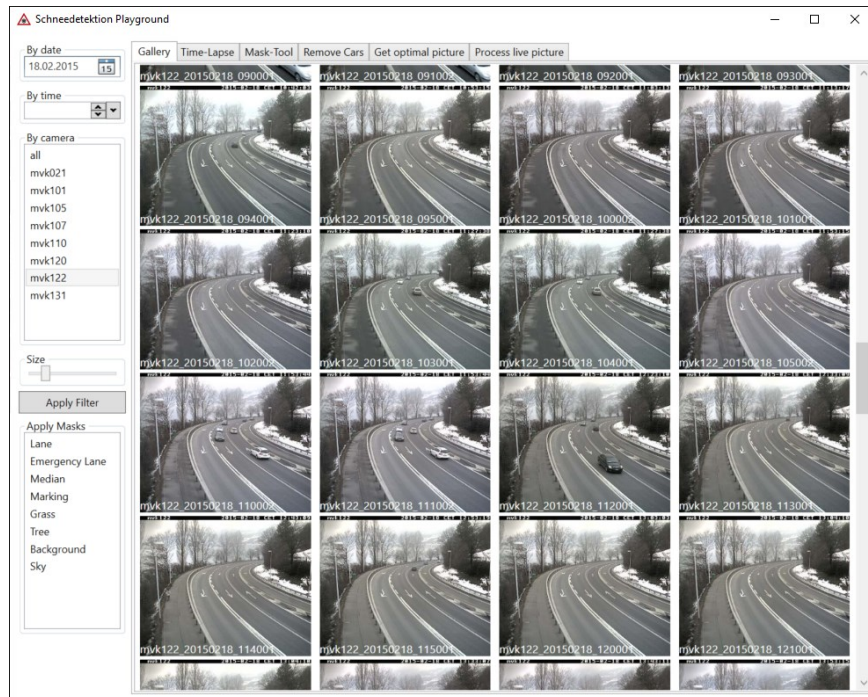
- Entwicklungsumgebung: Microsoft Visual Studio Community 2015
- Die Benutzerschnittstelle ist umgesetzt mit XAML (Windows Presentation Foundation)
- Für einige Komponenten der Benutzerschnittstelle wurde auf das Extended WPF Toolkit zurückgegriffen
- Die Programmlogik ist in C# der Version .NET 4.5.2 geschrieben
- Die Schnittstelle zur Datenbank bildet LINQ to SQL
- Für die Serialisierung und Deserialisierung von Objekten wurde Newtonsoft.Json verwendet
- Bei der Bildverarbeitung wird OpenCV genutzt. Das eine quelloffene und weit verbreitete Programmbibliothek für Computer Vision und Machine Learning.
- Da OpenCV keine Programmschnittstelle zu C# bereitstellt, werden OpenCV Operationen über eine weitere quelloffene Programmschnittstelle aufgerufen: Emgu CV
- Für Quellcodeverwaltung wurde GitHub gewählt. Hier ist der gesamte Programmcode, die Datenbank und die Dokumentation abgelegt und versioniert.

### 3.2 Metadaten in Datenbank

Um eine handhabbare Übersicht der bestehenden Daten zu schaffen, wurde eine Datenbank aufgebaut. Wegen der überschaubaren Datenmenge und der Möglichkeit der Persistierung in einer Quellcodeverwaltung, ist die Datenbank als lokale MDF-Datei realisiert. Diese Datei funktioniert ohne SQL-Server und ist direkt in die Entwicklungsumgebung eingebunden. Alle zugänglichen Daten über die Bilder konnten so in Tabellenform abgelegt werden. Zu jedem Bild sind folgende Werte pro Zeile abgespeichert: Dateipfad zu Bild, Kameraname und Zeitpunkt der Aufnahme.

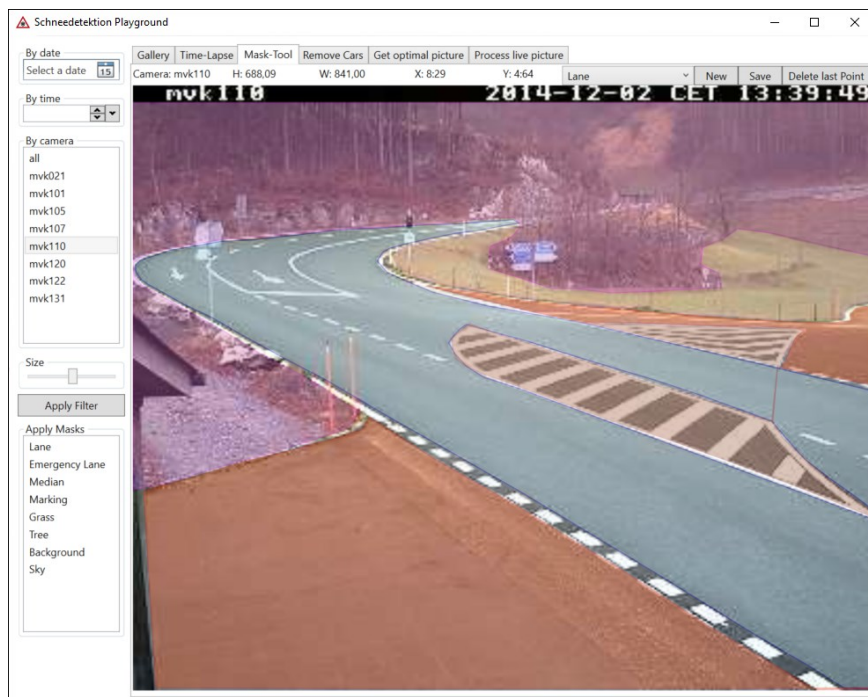
Für die Datenvisualisierung wurde eine Applikation entwickelt für die Betrachtung und Analyse der Bilder pro Kameraperspektive, Aufnahmezeitpunkt und Bildausschnitt:





### 3.3 Segmentierung des Bildes

Da der Schnee auf der Fahrbahn nicht lange liegen bleibt, sollte die Detektion sich auf die umliegenden Regionen der Strasse auf dem Bild konzentrieren. Hierzu wurde ein Werkzeug entwickelt, das ermöglicht, das Bild virtuell in verschiedene Regionen zu unterteilen. Zum Einsatz kamen folgende Bildregionen: Fahrbahn, Pannenstreifen, Mittelstreifen, Strassenmarkierung, Rasen, Hintergrund und Himmel. Zu jeder Kameraperspektive wurden die Regionen so definiert, dass möglichst jeder Bildpunkt zu genau einer Region gehört.



#### 3.3.1 Polygon Editor

Die Applikation wurde um einen einfachen Polygon-Editor erweitert. Dieser macht es möglich über das Bild per Mausklick farbige Polygone zu zeichnen. Über das die Darstellungsebene

wird eine virtuelle Zeichnungsebene gelegt. Auf diesem lassen sich die Polygone definieren. Jede Bildregion hat eine eigene Farbe:

- Fahrbahn: Blau
- Pannestreifen: Rot
- Mittelstreifen: Gelb
- Strassenmarkierung: Braun
- Rasen: Violett
- Hintergrund: Magenta
- Himmel: Gold

Per Drop-Down lässt sich die gewünschte Region auswählen. Die Zeichnungsebene verfolgt die Mausposition und bei jedem Klick wird die X- und Y-Position als der Maus auf der Ebene als neuer Punkt des Polygons hinzugefügt und dargestellt. Die definierten Polygone werden ebenfalls in die Datenbank abgespeichert. Davor werden die Positionen der Polygonpunkte auf zwischen 0 und 1 skaliert.

```
public static string SerializePointCollection(
    Polygon polygon, double viewWidth, double viewHeight)
{
    PointCollection pointCollection = new PointCollection();
    foreach (Point point in polygon.Points)
    {
        // Skalierung der Punkte zwischen 0 und 1
        // 1 / Fensterbreite * X-Position
        // 1 / Fensterhöhe * Y-Position
        pointCollection.Add(new Point(1 / viewWidth * point.X, 1 / viewHeight * point.Y));
    }
    // Serialisierung zu Json-Objekt
    return JsonConvert.SerializeObject(pointCollection);
}
```

Dies ist nötig, damit das Polygon nicht abhängig ist von der aktuellen Fenstergrösse der Applikation.

### 3.4 Bereinigung von Autos

Damit Autos, welche sich gerade auf der Aufnahme befinden, die Farben der Regionen nicht verfälschen, muss ein Weg gefunden werden die Fahrzeuge aus dem Bild zu entfernen. Die Idee ist hier, dass mehrere Bilder, die in kurzen Abständen aufgenommen wurden mit mathematischen Operationen miteinander verschmolzen werden. So soll ein neues, autofreies Bild entstehen, dass für die Weiterverarbeitung bestimmt ist.

#### 3.4.1 Algorithmus

Zunächst wird ein erstes Bild über die Web-Schnittstelle heruntergeladen und im Dateisystem abgelegt. Danach muss einige Sekunden gewartet werden. Falls in zu hoher Frequenz Bilder von der Webcam angefordert werden, ist es möglich, dass die Kamera dasselbe Bild noch einmal liefert. Anschliessend erfolgt die Berechnung der Differenz der beiden Bilder:

```
public void CalculateAbsoluteDifference
    (string imagePath0, string imagePath1, string resultPath)
{
    // Bilder einlesen
    Image<Bgr, byte> image0 = new Image<Bgr, byte>(imagePath0);
    Image<Bgr, byte> image1 = new Image<Bgr, byte>(imagePath1);

    // Neues, leeres Bild erzeugen
    Image<Bgr, byte> result1 = new Image<Bgr, byte>(new byte[288, 352, 3]);

    // Absolute Differenz der beiden Bilder berechnen
    result1 = image0.AbsDiff(image1);
    // Thresholden und Invertieren
    result1.ThresholdBinaryInv(new Bgr(50, 50, 50), new Bgr(255, 255, 255));
    // Schwarze Fläche grösser machen per Erosions-Operation
    result1._Erode(3);
}
```



```

    // Differenz der Bilder in neuem Bild abspeichern
    result1.Save(resultPath);
}

```

Das Resultat ist eine Bitmaske, welche schwarze Flecken enthält, dort wo sich die beiden Bilder unterscheiden. Die Bilder unterscheiden sich an jenen Stellen, wo ein Auto sich nur in einem der Bilder befindet.



Aus dem ersten Bild und der Bitmaske kann ein neues Bild (Kandidat) mit leeren, schwarzen Flecken erzeugt werden. Dies ist mit einem einfachen emguCV-Aufruf ausführbar:

```

public void GetMaskedImage (string imagePath, string maskPath, string resultPath)
{
    // Bilder einlesen
    Image<Bgr, byte> image = new Image<Bgr, byte>(imagePath);
    Image<Bgr, byte> mask = new Image<Bgr, byte>(maskPath);
    // Neues, leeres Bild erzeugen
    Image<Bgr, byte> result = new Image<Bgr, byte>(new byte[288, 352, 3]);

    // Bildpunkte in neues Bild kopieren, ausser dort wo Bitmaske Flecken hat
    CvInvoke.cvCopy(image, result, mask);

    result.Save(resultPath);
}

```

So sieht ein Kandidatenbild aus. Die Autos aus den Quell-Bildern sind entfernt. Es folgt der Versuch diese Löcher zu füllen.



Es kann kalkuliert werden, welcher Anteil der entstandenen Bitmaske, die zum Kandidaten gehört, schwarz ist. Falls mehr als ein Prozent der Bitmaske schwarz ist, wird ein weiteres Bild im Bereinigungsprozess berücksichtigt.

Erneut wird kurz gewartet und ein weiteres Bild von der Webcam heruntergeladen. Aus dem zweiten und dem dritten Bild lässt sich wieder die Bitmaske erzeugen, welche mit der vorangehenden Bitmaske verglichen wird. An denjenigen Stellen, wo sich die zwei Bitmasken unterscheiden, können nun Bildinhalte aus dem dritten Bild zum Kandidatenbild aufgefüllt werden:

```

public void FillMaskHoles(string maskPath0, string maskPath1,
    string newImagePath, string candidateImagePath, string resultPath)
{

```

```

// Masken und Bilder einlesen
Image<Bgr, byte> mask0 = new Image<Bgr, byte>(maskPath0);
Image<Bgr, byte> mask1 = new Image<Bgr, byte>(maskPath1);
Image<Bgr, byte> newImage = new Image<Bgr, byte>(newImagePath);
Image<Bgr, byte> candidate = new Image<Bgr, byte>(candidateImagePath);
// Neues, leeres Bild erzeugen
Image<Bgr, byte> resultMask = new Image<Bgr, byte>(new byte[288, 352, 3]);

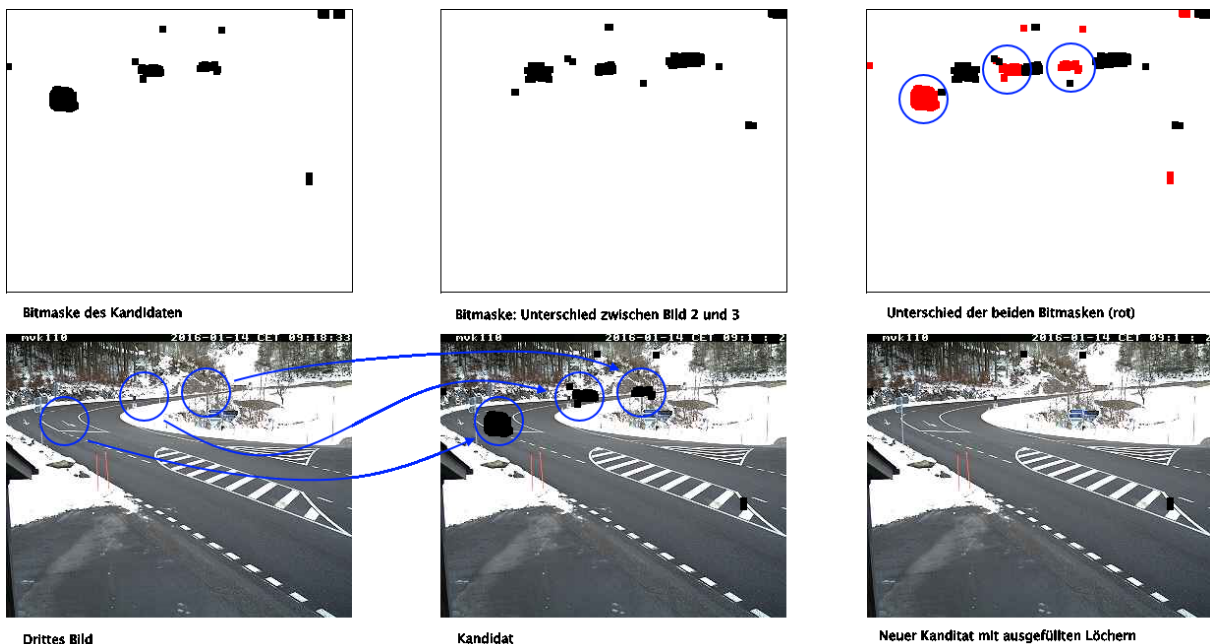
// Neue Maske erzeugen: Repräsentiert die Unterschiede der beiden Input-Masken
for (int i = 0; i < mask0.Cols; i++)
{
    for (int j = 0; j < mask0.Rows; j++)
    {
        if (mask0.Data[j, i, 0] == 0 &&
            mask0.Data[j, i, 1] == 0 &&
            mask0.Data[j, i, 2] == 0 &&
            mask1.Data[j, i, 0] == 255 &&
            mask1.Data[j, i, 1] == 255 &&
            mask1.Data[j, i, 2] == 255)
        {
            // Schwarzes Pixel setzen, wenn die Masken unterschiedlich sind
            resultMask.Data[j, i, 0] = 0;
            resultMask.Data[j, i, 1] = 0;
            resultMask.Data[j, i, 2] = 0;
        }
        else
        {
            // Weisses Pixel setzen, wenn die Masken gleich sind
            resultMask.Data[j, i, 0] = 255;
            resultMask.Data[j, i, 1] = 255;
            resultMask.Data[j, i, 2] = 255;
        }
    }
}
// Maske invertieren
resultMask._Not();

// Bildpunkte in neues Bild kopieren, ausser dort wo Bitmaske Flecken hat
CvInvoke.cvCopy(newImage, candidate, resultMask);

candidate.Save(resultPath);
}

```

So kann das Ausfüllen der Löcher im Kandidatenbild aussehen:



Dieser Prozess wird so lange wiederholt, bis die Bitmaske des Kandidatenbildes weniger als ein Prozent schwarz ist:

- Neues Bild herunterladen
- Differenz der letzten zwei Bildern erstellen (Bitmaske)
- Differenz zwischen dieser Bitmaske und der Kandidaten-Bitmaske erstellen
- Löcher im Kandidatenbild füllen mit Inhalt aus dem neuesten Bild
- Neue Kandidaten-Bitmaske erstellen
- Schwarzanteil der Kandidaten-Bitmaske berechnen

Nach der Bereinigung der Autos, wird das Bild in die vordefinierten Segmente auseinander geschnitten:

```
public BitmapImage GetPatchBitmapImage (string imagePath, IEnumerable<Point> polygon)
{
    // UMatrix aus Bild erstellen
    Mat matrix = new Mat(imagePath, LoadImageType.AnyColor);
    UMat uMatrix = matrix.ToUMat(AccessType.ReadWrite);

    // Polygon auf Bildgrösse skalieren
    List<Point> scaledPoints = GetScaledPoints(polygon, uMatrix.Cols, uMatrix.Rows);
    // Polygon invertieren. Es wird ein Polygon erstellt,
    // dass alles ausser die ursprüngliche Polygonfläche abdeckt
    List<Drawing.Point> polygonPoints =
        GetInvertedPoints(scaledPoints, uMatrix.Cols, uMatrix.Rows);

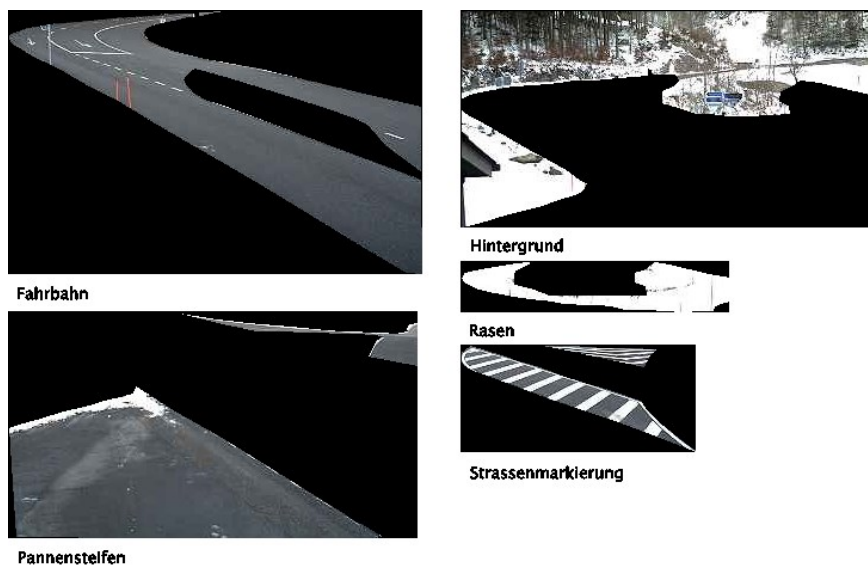
    // Invertiertes Polygon auf das Bild anwenden
    using (VectorOfPoint vPoint = new VectorOfPoint(polygonPoints.ToArray()))
    using (VectorOfVectorOfPoint vvPoint = new VectorOfVectorOfPoint(vPoint))
    {
        // Alles schwarz anmalen, dass nicht im invertierten Polygon ist
        CvInvoke.FillPoly(uMatrix, vvPoint, new Bgr(0, 0, 0).MCvScalar);
    }

    // Matrix zu Image umwandeln
    Image<Bgr, byte> image = new Image<Bgr, byte>(uMatrix.Bitmap);

    // Bild auf Polygon-Region beschneiden (Bounding-Box)
    // (schwarze Ränder oben, unten, links und rechts abschneiden)
    image.ROI = GetRegionOfInterest(scaledPoints);

    return OpenCVHelper.BitmapToBitmapImage(image.Bitmap);
}
```

Es resultieren von Autos bereinigte Bildsegmente für die Schneedetektion. Ausgehend vom oben bearbeiteten Beispiel entstehen fünf Segmente:



### 3.5 Naive Schneedetektion

Die Schneedetektion verfolgt nun einen naiven Ansatz: Für jede Region wird die Durchschnittsfarbe ausgerechnet. Dies bewerkstelligt ein einfacher OpenCV-Aufruf:

```
Bgr average = region.GetAverage(bitmask);
```

Der Rückgabewert ist eine OpenCV-Struktur (Bgr = Blue-Green-Red), dass den Durchschnittswert pro Kanal als Fließkommazahl enthält. Diese Struktur kann dreidimensionaler Vektor betrachtet werden. Die Distanz zu vorberechneten Referenzwerten errechnet sich mit der Euklidischen Distanz:

```
double distanceToSnow = Math.Sqrt(  
    Math.Pow(snow.Blue - average.Blue, 2) +  
    Math.Pow(snow.Green - average.Green, 2) +  
    Math.Pow(snow.Red - average.Red, 2));  
double distanceToNotSnow = Math.Sqrt(  
    Math.Pow(normal.Blue - average.Blue, 2) +  
    Math.Pow(normal.Green - average.Green, 2) +  
    Math.Pow(normal.Red - average.Red, 2));
```

Wenn die Durchschnittsfarbe von mehr als der Hälfte aller Regionen näher am Referenzwert für die Kategorie (Schnee oder kein Schnee) liegt, wird dem Bild die entsprechende Kategorie zugewiesen.

Da über das betrachtete Bild nichts über Licht- und Wetterverhältnisse bekannt ist, kann nur ein Vergleich zu globalen Referenzwerten gezogen werden. Für die Referenzwerte wurden für jede Kamera Bilder ausgewählt, welche die Kategorien 'es liegt Schnee' und 'es liegt kein Schnee' gut repräsentieren und nicht von Licht- und Wetterverhältnissen signifikant beeinflusst sind. Die Bilder wurden auf Regionen aufgetrennt. Pro Region konnte der Durchschnittswert ausgerechnet und in der Datenbank abgelegt werden.

Wenn eine Region positiv, dann ganzes Bild positiv

### 3.6 Resultate

Ursprünglich war nicht bekannt wie viel Zeit die Verarbeitung eines Bildes in Anspruch nimmt. Im Projektverlauf wurde jedoch klar, dass die Verarbeitungszeit nicht signifikant ist. Die Detektion für alle Kameras dauerte nur einige Sekunden und stellt für einen potentiellen Live-Betrieb keine Einschränkung dar.

Die Schneedetektion liefert nicht genügend gute Ergebnisse. Es gibt keine Möglichkeit die Referenzwerte dynamisch auf den Bildinhalt anzupassen. Häufig wird ein Bild zur Schnee-Kategorie zugewiesen, obwohl kein Schnee liegt, dafür beispielsweise Nebel die Umgebung viel weisser erscheinen lässt. Abgesehen von Nebel gibt es weitere Situationen in denen die Farben verfälscht sind:

- Dämmerung: Veränderte Farben, lange Schatten und direkte Sonneneinstrahlung in die Linse verändern die Farben aller Bildregionen
- Nachts wird nur ein schwarz-weisses Bild aufgenommen
- Bei Regenwetter können Scheinwerfer der Autos und die Sonne Reflektionen auf der nassen Fahrbahn entstehen lassen

Der Fokus in der Bachelorarbeit wurde aus diesen Gründen auf folgende Punkte gelegt:

- Referenzwerte für verschiedene Licht- und Wetterverhältnisse herstellen
- Systematisches testen des Algorithmus ermöglichen
- Eine Stichprobe erstellen, bei der zu jedem Bild die 'Wahrheit' bekannt ist: Zu jedem Bild Kategorien erfassen zu Schnee-, Licht- und Wetterverhältnissen und in der Datenbank abspeichern
- Weitere statistische Werte als nur die Durchschnittsfarbe einbeziehen wie Histogramm, Kontrast und Farbspektrum
- Konzepte aus Machine Learning für die Detektion verwenden

- Kleinere Regionen (Patches) betrachten und nicht das gesamte Bild in grosse Segmente einteilen. Für die ASTRA ist hauptsächlich interessant, ob unmittelbar neben der Strasse liegt. Es sollen Patches in Strassennähe betrachtet werden, auf welchen Schnee längere Zeit liegen bleibt.

### **3.7 Weitere Themen**

Im Verlauf der Projektarbeit wurden noch weitere Themen bearbeitet, welche schliesslich keine nennenswerten Resultate geliefert haben:

#### **3.7.1 Wetterdaten**

Einbezug von historischen und aktuellen Wetterdaten: Um dynamisch entscheiden zu können welche Licht- und Wetterverhältnisse auf dem Bild zu erwarten sind, sollen Wetterdaten von MeteoSchweiz berücksichtigt werden. Von Interesse wäre beispielsweise lokale Temperatur, Niederschlagsmenge und Luftfeuchtigkeit.

Leider betreibt Bundesamt für Meteorologie keine öffentliche Programmschnittstelle. Es existieren zwar öffentliche Schnittstellen (wie OpenWeatherMap.org), diese bieten Datenabfragen ab bestimmten Menge nur noch gegen ein Entgelt an. Die einzige Möglichkeit auf diese Daten zuzugreifen ist eine Applikation, die von der Abteilung für Extremwertanalyse der Fachhochschule entwickelt und betrieben wird. Diese bietet eine Exportfunktion aber keine Programmschnittstelle.

Ein beträchtlicher Teil der Projektzeit wurde in die Einarbeitung in die Applikation investiert. Aus Zeitgründen wurde das Thema aber nicht mehr weiterverfolgt.

#### **3.7.2 Intelligenter Cron-Job**

Der Cron-Job sollte so erweitert werden, dass der Bilddownload häufiger als alle zehn Minuten geschieht. Gleichzeitig soll ein Vorverarbeitungs-Job die Bilder bereits auf dem Server von Autos bereinigen. Dies wurde nicht weiterverfolgt, weil ich nicht genug Kenntnisse über Batch-Skripts habe.

## 4 Bachelorarbeit

### 4.1 Verbesserung zu Projekt 2

Im Projekt 2 konnte viel Wissen über den Datenstand und das Handwerk angeeignet werden. Nun aber sollen sich die Anstrengungen vermehrt auf die Verbesserung der Schneedetektion richten. Die Erkennungsrate des Algorithmus soll messbar sein und zuverlässiger sein als der naive Ansatz aus der Projektarbeit. Hierzu soll aus dem Bildarchiv Informationen gewonnen werden, die für den Computer verständlich sind:

Zu jedem Bild soll in der Datenbank gespeichert sein, ob neben der Strasse Schnee liegt oder nicht. Mit dieser Zusatzinformation kann die Güte eines neuen Algorithmus gemessen werden (F-Test). Weiter sollen auch Licht- und Wetterverhältnisse auf dem Bild bekannt sein. So wird es möglich, dass abhängig vom Bildinhalt gegen adäquate Referenzwerte geprüft wird.

### 4.2 F-Test

Um den Algorithmus systematisch testen zu können, kommt der F-Test zum Zug. Dieser Test kann eine Aussage über die Qualität eines Klassifikators machen. Dabei ist der Klassifikator ein Verfahren, das Objekte zu verschiedenen Klassen zuordnet. In unserem Fall sind gibt es die Klassen 'Schnee' und 'kein Schnee'.

Bei der Klassifizierung können Fehler passieren: Entweder wird einem Schnee-Bild die Kategorie 'kein Schnee' oder einem schneefreien Bild die Kategorie 'Schnee' zugewiesen. Daraus entstehen vier verschiedene Fälle:

	Es liegt Schnee	Es liegt kein Schnee
Test positiv	Richtig positiv ( $r_p$ )	Falsch positiv ( $f_p$ )
Test negativ	Falsch negativ ( $f_n$ )	Richtig negativ ( $r_n$ )

Nun können nach der Durchführung einer Klassifizierung des Datenstandes die richtig und falsch eingeordneten Bilder aufgezählt werden. Aus diesen Zahlen lassen sich drei Kennwerte berechnen:

$$\text{Sensitivität} = \frac{r_p}{r_p + f_n}$$

$$\text{Genauigkeit} = \frac{r_p}{r_p + f_p}$$

Die Sensitivität ist das Verhältnis zwischen den richtig detektierten Objekten zu allen positiv klassifizierten Objekten. Während die Genauigkeit ein Verhältnis zwischen den richtig detektierten und allen tatsächlich positiven Objekten ist. Aus diesen zwei Kennzahlen bildet sich der F-Wert:

$$F = 2 * \frac{\text{Sensitivität} * \text{Genauigkeit}}{\text{Sensitivität} + \text{Genauigkeit}}$$

Der Wertebereich aller drei Zahlen ist zwischen 0 und 1. Je höher die Werte sind, desto besser ist der Klassifikator.

### 4.3 Stichprobe aufbauen

Damit der Klassifikator prüfbar ist, muss also die Wahrheit bekannt sein. Das bedeutet zu möglichst jedem Bild aus dem Bildarchiv muss erfasst werden, ob auf dem Bild Schnee liegt oder nicht. Dafür wird die bestehende Applikation erweitert mit einer 'Klassifizierung per Hand'.



#### 4.3.1 Kategorien

Neben den zwei offensichtlichen Kategorien 'Schnee' und 'kein Schnee', gibt es weitere Fälle, bei denen der Bildinhalt beeinflusst ist. Abhängig vom Sonnenstand und den Wetterverhältnissen verändern sich die abgebildeten Farben und Sichtverhältnisse. Es kamen insgesamt folgende Bild-Kategorien zum Vorschein:



Kein Schnee



Schnee



Tag



Dämmerung  
(Morgen oder Abend)



Nacht



Nebel



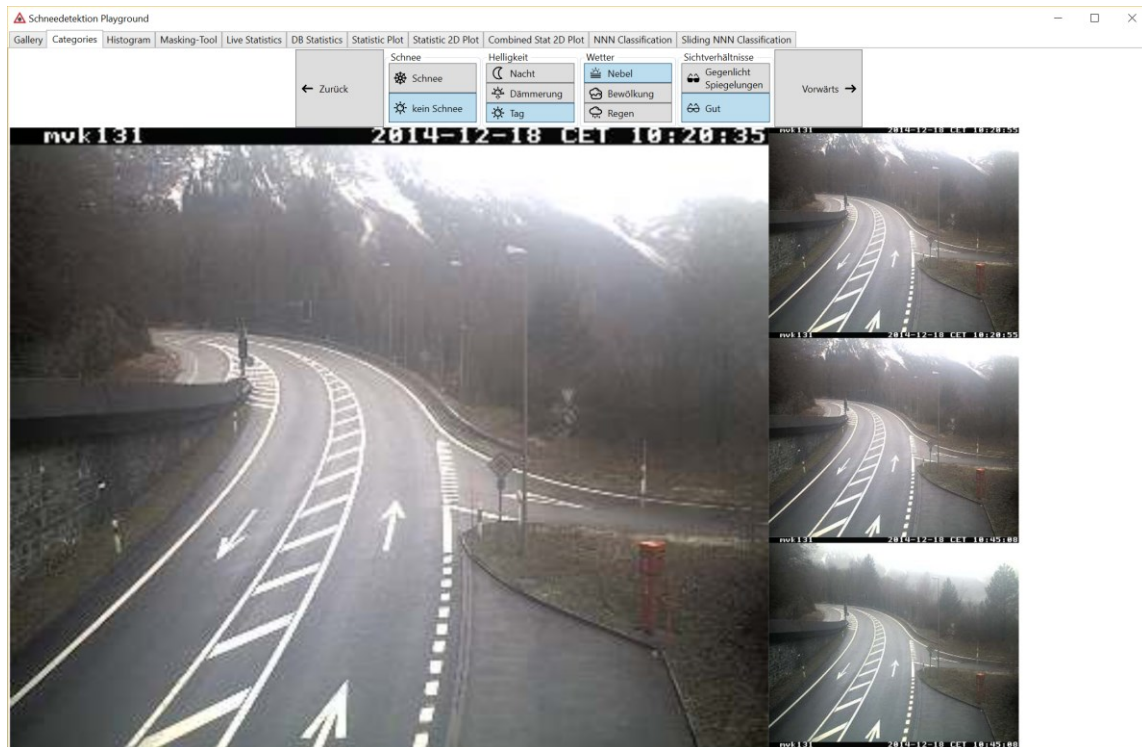
Niederschlag



Reflektionen oder direkte  
Sonneneinstrahlung

Zudem können auch mehrere Beeinträchtigungen gleichzeitig eintreffen. So können zum Beispiel Scheinwerfer auf einer regennassen Strasse Reflektionen verursachen oder Sonneneinstrahlung bei Nebel am Morgen das Bild heller erscheinen lassen. Dies macht die naive Detektion mit dem Vergleich zu globalen Referenzwerten zu fehleranfällig.

## 4.3.2 Programm



Um den Aufbau der Stichprobe in einer möglichst kurzen Zeit zu bewältigen wurde die Applikation um eine 'Kategorisierungs-Modul' erweitert. In diesem Modul werden Bilder in ihrer Aufnahmereihenfolge geladen und eines nach dem anderen gross angezeigt. Daneben werden einige nachfolgende Bilder angezeigt, die einen Überblick geben, wie sich die Sichtverhältnisse in den darauffolgenden Minuten verändert haben.

Mit den Knöpfen oberhalb der Bilder lässt sich die Situation im Bild auf die Kategorien verteilen und das Bild wechseln. Um die Bilder schneller durcharbeiten zu können ist das Modul vollständig mit der Tastatur bedienbar:

- Vorwärts und Zurück auf der Zeitachse ist auf die rechte beziehungsweise linke Pfeiltaste abgebildet. Bei jedem Wechsel auf das nächste oder vorangehende Bild, wird der definierte Zustand zum Datensatz gespeichert.
- Nummerntaste 1 schaltet zwischen 'Schnee' und 'kein Schnee' um
- Nummerntaste 2 wechselt periodisch die Tageszeitoptionen:  
Nacht => Dämmerung => Tag => Dämmerung => Nacht => ...
- Nummerntasten 3, 4 und 5 aktivieren und deaktivieren die Nebel-, Bewölkungs- und Regen-Option
- Nummerntaste 6 schaltet zwischen guten und schlechten Lichtverhältnissen um

Auf Datenbankebene wurde die Bild-Tabelle so erweitert, dass neben dem Dateipfad, der Kamera und dem Aufnahmezeitpunkt auch die Kategoriezugehörigkeit als Wahrheitswert (Boolean / Bit) gespeichert werden kann:

```
CREATE TABLE [dbo].[Images] (
    [ID] INT IDENTITY (1, 1) NOT NULL,
    [Name] NVARCHAR (50) NOT NULL,
    [Place] NVARCHAR (50) NOT NULL,
    [DateTime] DATETIME NOT NULL,
    [TimeZone] NVARCHAR (5) NULL,
    [UnixTime] FLOAT (53) NULL,
    [Snow] BIT NULL,
    [NoSnow] BIT NULL,
    [Night] BIT NULL,
    [Dusk] BIT NULL,
    [Day] BIT NULL,
```

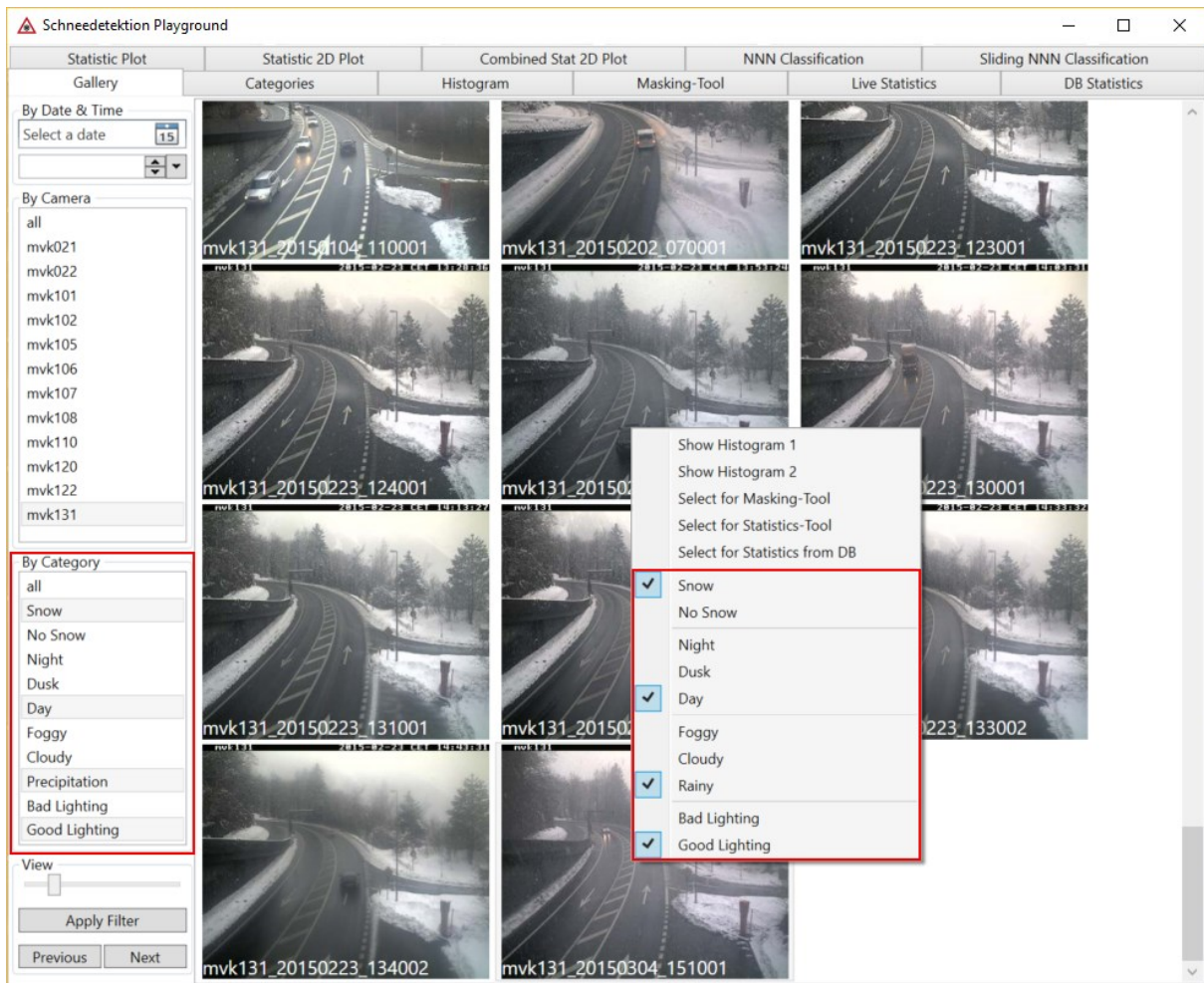
```

[Foggy]          BIT          NULL,
[Cloudy]         BIT          NULL,
[Rainy]          BIT          NULL,
[BadLighting]    BIT          NULL,
[GoodLighting]   BIT          NULL,
PRIMARY KEY CLUSTERED ([ID] ASC)
);

```

#### 4.3.3 Anpassungen Galerie Modul

Für die stichprobenartige Kontrolle und allfällige Korrektur der gespeicherten Kategorien wurde die Galerie-Modul um zwei Funktionen erweitert: Ein Kontextmenu welches die zugewiesenen Kategorien anzeigt und anpassbar macht und ein Filter mit dem sich die Bildliste neben Kamera und Datum auch auf Kategorien einschränken lässt



#### 4.3.4 Erfassung der Kategorien

Das Kategorisierungs-Modul ermöglichte es die ca. 400'000 Bilder zügig durchzuarbeiten. Im Schnitt wurde pro Bild weniger als eine halbe Sekunde aufgewendet. Das bedeutet, dass die Erfassung der Kategorien für das gesamte Bildarchiv etwa 45 Arbeitsstunden im Oktober in Anspruch nahm.

#### 4.3.5 Verhältnis

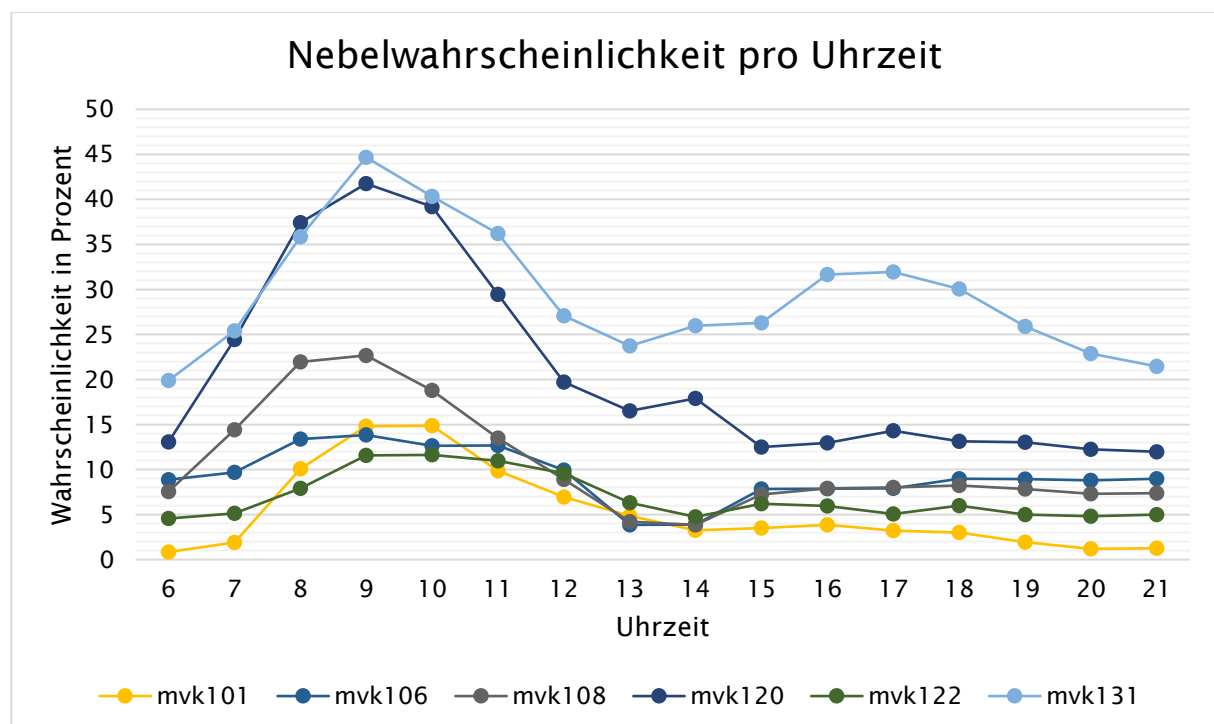
Aus den nun verfügbaren Daten können nun die ersten einfachen statistischen Analysen erstellt werden. So kann ein Überblick über die absolute Anzahl von Schnee Bildern und das Verhältnis zu den schneefreien Aufnahmen erstellt werden:

Kamera	Bilder insgesamt	Bilder ohne Schnee	Anteil ohne Schnee (%)	Bilder mit Schnee	Anteil mit Schnee (%)
mvk021	37894	32264	85,14	5630	14,86
mvk022	15760	14020	88,96	1740	11,04
mvk101	34145	29396	86,09	4749	13,91
mvk102	19377	18344	94,67	1033	5,33
mvk105	27815	24385	87,67	3430	12,33
mvk106	21100	19599	92,89	1501	7,11
mvk107	31949	28077	87,88	3872	12,12
mvk108	21114	19148	90,69	1966	9,31
mvk110	51763	30066	58,08	21697	41,92
mvk120	54118	34734	64,18	19384	35,82
mvk122	51232	43172	84,27	8060	15,73
mvk131	51844	29166	56,26	22678	43,74

Es ist sichtbar, dass für drei Kameras (mvk110, mvk120, mvk131) das Schneeverhältnis deutlich höher ist als bei den anderen. Diese Kameras befinden sich auf der A8, die Strasse zwischen Brienersee und Brünigpass Passhöhe. Diese befinden sich auf einer höheren Lage und in einer Region, in welcher hohe Berge im Tagesverlauf länger Schatten auf die Strasse werfen. Hier bleibt der Schnee länger liegen.

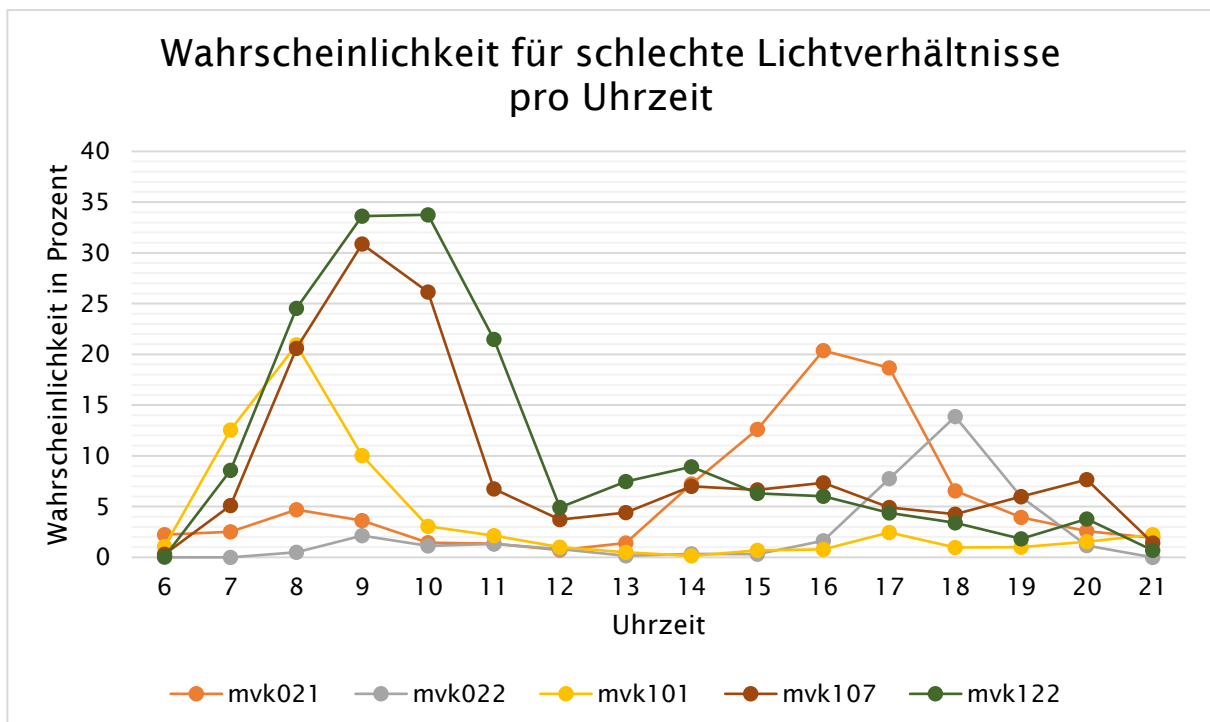
#### 4.3.6 Wahrscheinlichkeit für eingeschränkte Sichtverhältnisse

Beim Erfassen der Kategorien ist aufgefallen, dass eingeschränkte Sichtverhältnisse pro Kamera immer wieder zu gleichen Uhrzeiten auftreten. Dies bestätigt sich bei genauerer Betrachtung der Zahlen:



In der Winterzeit ist die Wahrscheinlichkeit für eine Nebellage für alle Kameras am höchsten zwischen 8 und 10 Uhr morgens. Hier stehen zwei Kameras (mvk120 und mvk131) auf der A8-Route besonders heraus: Hier ist die Wahrscheinlichkeit für Nebel um 9 Uhr morgens mit über 40 Prozent am höchsten.





Abhängig davon wie die Kamera platziert ist die Wahrscheinlichkeit für schlechte Lichtverhältnisse ebenfalls immer zu gleichen Uhrzeiten höher: Kameras die nach Osten ausgerichtet sind und teilweise den Himmel im Bild haben, sind am Morgen von der direkten Sonneneinstrahlung beeinflusst (mvk101, mvk107 und mvk122). Dasselbe gilt für Kameras mit Ausrichtung nach Westen – nur scheint da die Abendsonne in die Kamera (mvk021 und mvk022).

#### 4.4 Neue Segmentierung (Patches)

Im Verlauf der Projekt 2 stellte sich heraus, dass es für die Detektion keinen Sinn macht das gesamte Bild in grosse Segmente zu unterteilen. Die farbliche Variation in den Segmenten ist viel zu gross, als dass der Unterschied zwischen schneefrei und liegendem Schnee zu gering ist für die Detektion. Das Interesse der Auftraggeber konzentriert sich auf die Information, ob unmittelbar neben der Strasse Schnee liegt oder nicht. Aus der näheren Betrachtung fällt darum die Fahrbahn heraus – hier bleibt kein Schnee liegen. Die Räumfahrzeuge säubern schneebedeckte Strassen innerhalb von Stunden. Auch Hintergrund des Bildes und der Himmel sind für eine Detektion nicht nützlich.

Die Regionen wurden mit dem bestehenden Polygon-Editor neu definiert und fortan als Patches (Flickstellen) bezeichnet. Diese decken Regionen ab, auf denen Schnee liegen nach dem Schneefall bleibt: Mittel-, Pannenstreifen und Rasenflächen neben der Fahrbahn.



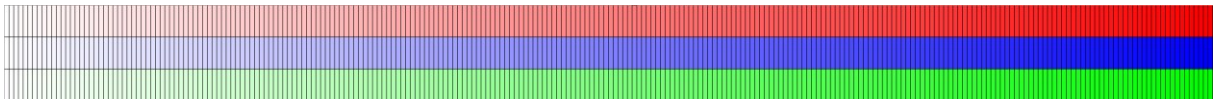
Für die Kamera mvk101 wurden fünf beispielsweise Patches definiert: Drei Rasenflächen um die Autobahneinfahrt, ein Stück zwischen Leitplanken und ein Strassenstück mit Markierungen.

## 4.5 Statistische Werte pro Patch

Eine weitere Lehre aus der Projektarbeit war, dass allein der Durchschnittsfarbwert für die Detektion nicht ausreicht. Aus den Bildern und Bildausschnitten lassen sich noch viele weitere Informationen extrahieren. Hierzu unterziehen wir die Bilder einer statistischen Auswertung.

### 4.5.1 Histogramm

Die zu analysierenden Bilder im Archiv sind normale JPEG-Dateien mit einer Farbtiefe von 24 Bit. Das bedeutet, dass Jeder Bildpunkt aus einer roten, blauen und grünen Farbkomponente besteht und jede Komponentenwert in 8 Bit codiert ist. Jede Farbkomponente kann Werte zwischen 0 und 255 annehmen. Je höher der Wert desto intensiver ist die Farbe. Die Tabelle unten zeigt alle Farben pro Kanal, die eine 8-Bit-Farbe annehmen kann:



Das Histogramm ist eine diskrete Häufigkeitsverteilung von Farbwerten in einem Bild. Um das Histogramm eines Bildes zu generieren, muss pro Bildpunkt und pro Farbkanal aufgezählt werden, wie häufig jeder der 256 verschiedenen Werte im Bild vorkommt.

```
private Statistic GetStatistic(Image<Bgr, byte> image, Image<Gray, byte> bitmask)
{
    // Pro Kanal eine leere Liste der Länge 256 erstellen
    List<double> blueHistogram = new List<double>();
    blueHistogram.AddRange(new double[256]);
    List<double> greenHistogram = new List<double>();
    greenHistogram.AddRange(new double[256]);
    List<double> redHistogram = new List<double>();
    redHistogram.AddRange(new double[256]);

    // Für alle Spalten im Bild
    for (int i = 0; i < image.Cols; i++)
    {
        // Für alle Zeilen im Bild
        for (int j = 0; j < image.Rows; j++)
        {
            // Wenn keine Bitmaske angegeben wurde,
            // oder die Bitmaske an dieser stelle schwarz ist
            if (bitmask == null || bitmask[j, i].MCvScalar.V0 == 0)
            {
                // Histogram pro Kanal abfüllen:
                // Listen-Element inkrementieren,
                // dass dem Farbwert des aktuellen Bildpunkts entspricht
                blueHistogram[(int)image[j, i].MCvScalar.V0]++;
                greenHistogram[(int)image[j, i].MCvScalar.V1]++;
                redHistogram[(int)image[j, i].MCvScalar.V2]++;
            }
        }
    }
    [...]
}
```



#### **4.5.2 Mode**

#### **4.5.3 Mean**

#### **4.5.4 Median**

#### **4.5.5 Minimum**

#### **4.5.6 Maximum**

#### **4.5.7 Standard Abweichung**

#### **4.5.8 Varianz**

#### **4.5.9 Kontrast**

#### **4.5.10 Werte ohne Hilfe von openCV berechnet**

#### **4.5.11 Patch Crop ohne schwarzen Rand**

### **4.6 Kombinieren**

#### **4.6.1 2h Slots pro Woche kombinieren und abspeichern**

#### **4.6.2 Mean**

#### **4.6.3 Median**

### **4.7 Plots**

#### **4.7.1 Clickable**

#### **4.7.2 Combined Plots**

### **4.8 Klassifizierer**

#### **4.8.1 NNN**

#### **4.8.2 Optimierungen**

##### **4.8.2.1 Input Kombinieren**

##### **4.8.2.2 Abstand ausrechnen aus 3 Inputs und entfernten verwerfen**

#### **4.8.3 F-Werte**

#### **4.8.4 Echte Verhältnisse**

#### **4.8.5 Ergebnisse**

#### **4.9 Weitere Themen**

##### **4.9.1 Sonneneinstrahlungs-Patch**

##### **4.9.2 Logit Regression mit Daniel Bättig**

##### **4.9.3 Albedo-Wert**

#### **4.10 OneDrive Probleme**

##### **4.10.1 Anderer Computer**

##### **4.10.2 Zugriff**

##### **4.10.3 Backups**

##### **4.10.4 Azure**

## 5 Schlussfolgerungen/Fazit

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit.

## 6 Abbildungsverzeichnis

Abbildung 1: Et ut aut isti repuditis qui ium

Error! Bookmark not defined.

## 7 Tabellenverzeichnis

Tabelle 1: Et ut aut isti repuditis qui ium

Error! Bookmark not defined.

## 8 Glossar

### **Auinweon**

Et ut aut isti repuditis qui ium	7
----------------------------------	---

### **Batnwpe**

Et ut aut isti repuditis qui ium	9
----------------------------------	---

### **Cowoll**

Et ut aut isti repuditis qui ium	11
----------------------------------	----

## 9 Literaturverzeichnis

### **Literatureintrag**

<i>Autorname, Autorvorname, Buchtitel, Verlag, Ort, Ausgabe, Jahr</i>	7
---	---

### **Literatureintrag**

<i>Autorname, Autorvorname, Buchtitel, Verlag, Ort, Ausgabe, Jahr</i>	9
---	---

### **Literatureintrag**

<i>Autorname, Autorvorname, Buchtitel, Verlag, Ort, Ausgabe, Jahr</i>	11
---	----

## 10Anhang

Et ut aut isti repuditis qui ium nonsecturia quis incientiae laborem elli quis et quatur, sitiur aut od moluptatur aut ea consequere peri sim erro essequisit remporia dem et landi dest, cone poris quunt volecab ipidero quatur ad quibusamus.

## 11 Selbständigkeitserklärung

Ich bestätige, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt habe. Sämtliche Textstellen, die nicht von mir stammen, sind als Zitate gekennzeichnet und mit dem genauen Hinweis auf ihre Herkunft versehen.

Ort, Datum:

Unterschrift: