

STARHack Asimov

Dokumentation

David Schafer, Serge Hauri, Steve Ineichen, Remo Schwarzenruber

2019-03-24

Inhaltsverzeichnis

1 Management Summary	4
1.1 Inspiration	4
1.2 Um was geht es?	4
1.3 Wie wurde es umgesetzt?	4
1.4 Herausforderungen	4
1.5 Accomplishments	4
1.6 Was wir gelernt haben	5
1.7 What's next?	5
1.8 Built with <3	5
1.9 Github Repo	5
2 Getting Started	6
2.1 Challenges	6
2.2 AUTONSENSE / VOLVO Challenge	6
2.3 Setup	7
3 Tasks	8
3.1 Arbeitspakete	8
3.2 Zuteilung der Arbeitspakete	8
4 Data Parser / Data Processing	9
4.1 Funktionsumfang	9
4.1.1 Software Abhängigkeiten	9
4.1.2 Prozess des Funktionsdesigns	9
4.2 Realisation / Umsetzung	9
4.3 Klasse DamageImage	9
4.3.1 Methode: parse_input_data	10
4.3.2 Methode: get_rel_times	10
4.3.3 Methode: __base64_decode	11
4.3.4 Methode: __convert_timestamps	11
4.3.5 Methode: calibrate_impact_data	11
4.3.6 Methode: __norm_with_g	12
4.3.7 Methode: __calculate_forces	12
4.3.8 Methode: __calculate_max_force	12
4.3.9 Methode: __calculate_offset_max_force	12
4.3.10 Methode: __calculate_custom_offset_force	13
4.3.11 Methode: __calculate_angle	13
4.3.12 Methode: __ringbuffer2array	13
4.3.13 Methode: __read_json_from_filesystem	13
4.3.14 Methode: __get_b64payload_from_basejson	14
4.3.15 Methode: __encoded_payload_to_list	14
4.4 Implementierung im Projekt	14
5 Damage drawer	15
5.1 Funktionsumfang	15

5.2	Funktiondesign	15
5.2.1	Software Abhängigkeiten	15
5.2.2	Prozess des Funktiondesigns	15
5.3	Realisation / Umsetzung	16
5.4	Klasse DamageImage	17
5.4.1	Methode: Init	17
5.4.2	Methode: Kulturen Auto	17
5.4.3	Methode: Zeichnen	18
5.4.4	Methode: Zeichnen - Pfeil	18
5.4.5	Methode: Zeichnen - Kreis	18
5.4.6	Methode: Text auf das Bild	18
5.4.7	Methode: Berechnung Beschädigung	19
5.4.8	Methode: Schreiben der Bilddatei	19
5.4.9	Methode: Pfad der geschriebene Datei	19
5.4.10	Methode: Löschen von allen gerendert Dateien	20
5.4.11	Methode: Anzeige der Datei auf dem Bildschirm	20
5.5	Implementierung im Projekt	20
5.6	Mögliche Darstellung der Datei in einem Portal	21

6 Server / Frontent und Backend

22

1 Management Summary

1.1 Inspiration

Die Challenge von autoSense (autoSense ist eine Tochtergesellschaft der Swisscom AG) hatte wir bereits ein breites Wissen von Teilaufgaben, wie Physik, Visualisierung, Präsentation und Kommunikation durch Webtechnologien. Das war dann auch unsere Motivation, uns in dieser Challenge zu messen.

1.2 Um was geht es?

Die Lösung empfängt json-Dateien durch ein ansprechendes Web-UI und verarbeitet diese. Als Ergebnis soll die grössen Beschädigung die auf das Auto eingewirkt hat, visualisiert werden. Die Visualisierung soll noch mit zusätzliche Informationen wie G-Force, Zeitpunkt des Crashes und den Offset in der Zeitreihe angereichert werden.

1.3 Wie wurde es umgesetzt?

Als Fundament haben wir einen Sanic v18 Webserver verwendet, um die API-Aufrufe zu implementieren. Im Hintergrund sind alle logischen Funktionen und Klassen mit Python sauber implementiert. Für den Visualisierungsteil haben wir uns für OpenCV 4.0 entschieden. Wir haben das gesamte Projekt auch als Docker Container umgesetzt, so dass es überall und jederzeit eingesetzt werden kann.

Um dem Projekt einige Extras hinzuzufügen, haben wir uns entschieden, ein kleines, übersichtliches WebUI zu erstellen, in dem wir grundlegende HTML5, CSS und natürlich Javascript verwendet haben.

1.4 Herausforderungen

Die Physik, um den Aufprall zu berechnen, hat definitiv sehr viel Zeit in Anspruch genommen. Wir haben manchmal einige komische Werte bekommen - dass war zum Teil darum, weil wir auf ein Transformationsproblem gestossen sind. Nach der Behebung von diesen Problemen sahen dann die Visualisierungen einheitlicher aus.

1.5 Accomplishments

Wir sind stolz darauf: * Solide Mathematik * maintainable Code * Clean Code * Visualisierung

1.6 Was wir gelernt haben

Wir haben viel Mathematik/Physik angewendet und natürlich OpenCV, was wir in der Schule einmal gelernt haben aber zum Teil wieder vergessen wurde. Der Hackathon war für uns als Gruppe ein wirklich gutes Training, wir haben gelernt, wie man besser und effizienter miteinander zusammenarbeitet, um ein grösseres Projekt wie die autoSense-Challenge anzugehen.

1.7 What's next?

Als nächstes wollen wir unsere Lösung aufpolieren, auch würden wir gerne noch ein paar Crash-Daten und ein paar Realbilder bekommen, damit wir überprüfen können, ob wir die richtigen Dinge berechnet haben. Außerdem gibt es einen Azure Webspace, der darauf wartet, dass unsere App veröffentlicht wird.

1.8 Built with <3

- python
- docker
- vanillajs
- html5
- css
- azure
- opencv
- sanic
- javascript

1.9 Github Repo

[Github Repo](#)

2 Getting Started

2.1 Challenges

There were 8 different challenges which you could apply. We were mainly interested in the Challenges from the following partners:

- Autosense (Crash Visualization)
- SBB (Recycle)
- Laica (AR)
- BOSCH IOT-Lab (Sensor Car)

All case descriptions can be viewed here: <http://live.starthack.ch/case-descriptions/>

We applied for the Autosense challenge and got it (limit of 15 Teams per challenge).
The challenge is as follows:

2.2 AUTONSENSE / VOLVO Challenge

Generate Car Crash Image, visualize impact and direction using sensor data

Your challenge if you choose to accept:

Build Microservice(s) to generate Image with 3D object simulating impact forces for given time offset (from crash). Deploy Microservice(s) on Swisscom Application Cloud (cloud foundry). Provide API(s) for submitting Input data (stream) and getting the Result. Generate output for each submitted Crash Record : Direction of the impact (Impact angle and energy), visualize the damage show expected place of impact on car

Winner is the Team who:

Has identified the maximum number of crashes correctly providing - Correct impact direction & Most accurate 3D simulation (compared to real crash picture)

How it will be measured:

For each submitted Crash Record AND time offset, generate Image with Direction of the impact (Impact angle and energy), Visualized damage and Time offset with the maximum force/damage on the object. Crash Record is submitted to the service. The calculated impact direction will be compared with pictures from real crash.

Restrictions:

Service must be deployable on cloud infrastructure (AWS/Cloud Foundry/Kubernetes/Docker). Service should use as few as possible external APIs. Given Data Models and API POST Requests structure must be used.

2.3 Setup

xxxxx

pwd

3 Tasks

3.1 Arbeitspakete

Um die Arbeiten besser auf die jeweiligen Team-Mitglieder aufzuteilen haben wir am Freitag Abend folgende Arbeitspakete definiert. *Die Arbeitspakete wurde 1:1 von dem Projekt-Meeting am Freitag Abend übernommen*

- * Data Parsing (transform in more structured way -> acceleration, calibration)
 - * define useful functions
 - * implement functions
 - * crash_record.py
- * Webserver
 - * create webserver (sanic)
 - * implement requests
 - * return some dummy data for the moment
 - * webserver.py (rename main.py)
 - * docker container
- * Image
 - * define interface
 - * library to draw arrows
 - * library to draw circles
 - * image.py
- * Visualization & Math
 - * jupyter notebook visualization
 - * define functions to calculate angles & impact
 - * start crash_record_calculator.py

3.2 Zuteilung der Arbeitspakete

Nach dem wir die AP (Arbeitspakete) definiert haben, konnte jeder sein präferiertes AP wählen. Die Aufteilung hat bei uns sehr gut funktioniert und wir konnten auf Anhieb jedes AP zu einer Person zuweisen.

4 Data Parser / Data Processing

4.1 Funktionsumfang

Mit der Klasse `Data_Parser` werden alle Funktionalitäten im Zusammenhang mit der Datenverarbeitung, AUSwertung und Konvertierung erledigt.

Das beinhaltet das einlesen der JSON Daten, ausfiltern der relevanten Key:Value Paare, transformieren der relativen Werte sowie die mathematischen Umrechnungen auf die geforderten Output Daten (Kraft & Winkel des Einschlags)

4.1.1 Software Abhängigkeiten

Zum auslesen der JSON Daten wurde die Python Library `json` verwendet. Zur Decodierung und Encodierung wurde das `base64` Format verwendet. Für die Mathematischen umrechnungen wurden die Libraries `math`, `pandas` sowie `numpy` verwendet.

Für die Umrechnung der relativen Zeiten konnte auf die Standard Library `'datetime'` zurückgegriffen werden.

Um während dem Testing ein sauberes Logging zu erhalten, wurde auch hier unsere gemeinsame Log Klasse `log_helper` eingebunden.

4.1.2 Prozess des Funktiondesigns

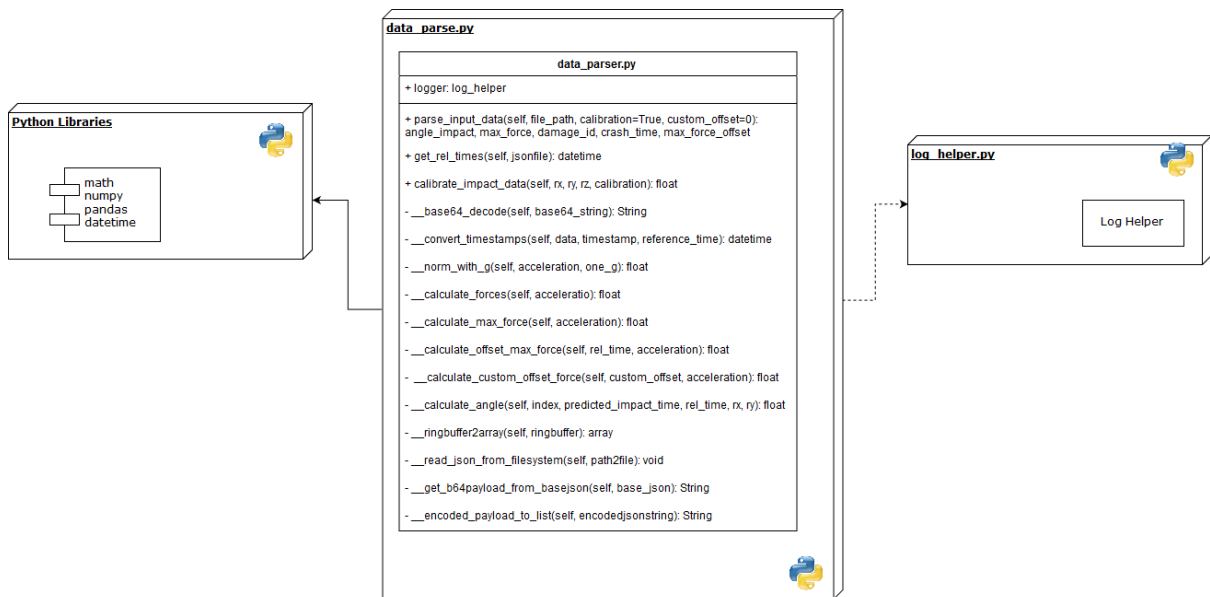
Zu Beginn werden die JSON Input Daten eingelesen und der Payload extrahiert. dieser wird mit `base64` decodiert und danach die relevanten Key:Value Paare (Beuschlungung `x,y,z`) ausgelesen. Diese Daten werden danach mit einer vordefinierten Kalibration transformiert und mit einem Referenz-G Wert in G-Kräfte umgewandelt. Zum Schluss erfolgt die Umwandlung der Kräfte in einen Winkel, relativ zur virtuellen Bildmitte. Zeit, Kraft und Winkel werden danach zur Visualisierung an die Klasse `Damage_drawer` übergeben.

4.2 Realisation / Umsetzung

Klassendiagramm und Beschreibung der Funktionen der einzelnen Methoden.

4.3 Klasse `DamageImage`

Teilfunktion: `Damage Image`



4.3.1 Methode: parse_input_data

```
def parse_input_data(self, file_path, calibration=True, custom_offset=0):
```

Parameter	Beschreibung
self	Instanz-Referenz
file_path	Das Input JSON File, entweder als Pfad zum Filesystem oder direkt als Objekt
calibration	Angabe ob die Werte mit oder ohne Kalibration berechnet werden sollen
custom_offset	Spezifische Offsettime (im Range 0 - 16000). Ohne Angabe wird der Offset mit der grössten G-Kraft verwendet

Die Methode `parse_input_data` wird von extern verwendet und führt alle Teilfunktionen zusammen. Das JSON File wird entweder als Objekt oder als Filepath übergeben und danach verarbeitet. Zusätzlich besteht die Möglichkeit die Berechnungen ohne Kalibration auszuführen (nur für Testzwecke nützlich). Ausserdem kann via `custom_offset` ein beliebiger Offset in Millisekunden angegeben werden, standardmässig wird der Zeitpunkt der grössten Krafteinwirkung selbst berechnet.

Als Rückgabeparameter liefert die Methode:

Return	Beschreibung
angle_impact	Einschlagswinkel
max_force	Die maximale G-Kraft
damage_id	Die Crash ID aus dem JSON File
crash_time	Die Genaue Zeit des Einschlags mit Maximaler Kraft
max_force_offset	Den Offset zum Beginn des Einschlags bis Maximal Kraft erreicht wurde

4.3.2 Methode: get_rel_times

```
def get_rel_times(self, jsonfile):
```

Parameter	Beschreibung
<code>self</code>	Instanz-Referenz
<code>jsonfile</code>	Das Input JSON File

Die Methode `get_rel_times` wandelt die Relativen Zeiten Anhand des Referenzwertes zu realen Zeiten um. Diese Information wird benötigt um den genauen Zeitpunkt zurückzugeben, an dem am meisten Kräfte auf das Auto wirkten.

4.3.3 Methode: `__base64_decode`

```
def __base64_decode(self, base64_string):
```

Parameter	Beschreibung
<code>self</code>	Instanz-Referenz
<code>base64_string</code>	Base64 Payload als String

Die Methode `__base64_decode` wandelt einen base64 Payload in einen UTF-8 json Payload um.

4.3.4 Methode: `__convert_timestamps`

```
def __convert_timestamps(self, data, timestamp, reference_time):
```

Parameter	Beschreibung
<code>self</code>	Instanz-Referenz
<code>data</code>	Data Payload (JSON)
<code>timestamp</code>	Den Timestamp aus dem Input JSON
<code>reference_time</code>	Die Referenzzeit aus dem Input JSON

Die Methode `__convert_timestamps` wandelt die relativen Zeiten der einzelnen Beschleunigungsmessungen in reale Zeiten um.

4.3.5 Methode: `calibrate_impact_data`

```
def calibrate_impact_data(self, rx, ry, rz, calibration):
```

Parameter	Beschreibung
<code>self</code>	Instanz-Referenz
<code>rx</code>	Array der X-Achsen Beschleunigungen
<code>ry</code>	Array der Y-Achsen Beschleunigungen
<code>rz</code>	Array der Z-Achsen Beschleunigungen
<code>calibration</code>	Referenz Kalibration von Autosense

Die Methode `calibrate_impact_data` wandelt die Arrays der x,y und z Beschleunigungen mit der von Autosense vorgegebenen Kalibration um, somit erhält man die Beschleunigungsvektoren rela-

tiv zur Auto Mitte.

4.3.6 Methode: `__norm_with_g`

```
def __norm_with_g(self, acceleration, one_g):
```

Parameter	Beschreibung
self	Instanz-Referenz
acceleration	Beschleunigungsvektor
one_g	Referenz G Kraft

Die Methode `__norm_with_g` berechnet anhand der vorgegebenen G-Kraft die normierte G-Kraft des Beschleunigungsvektors.

4.3.7 Methode: `__calculate_forces`

```
def __calculate_forces(self, acceleration):
```

Parameter	Beschreibung
self	Instanz-Referenz
acceleration	Beschleunigungsvektor

Die Methode `__calculate_forces` berechnet mittels Wurzelrechnung die effektiven Kräfte.

4.3.8 Methode: `__calculate_max_force`

```
def __calculate_max_force(self, acceleration):
```

Parameter	Beschreibung
self	Instanz-Referenz
acceleration	Beschleunigungsvektor

Die Methode `__calculate_max_force` berechnet, wann die grösste Kraft in der Messung auftrat und gibt diese Kraft zurück.

4.3.9 Methode: `__calculate_offset_max_force`

```
def __calculate_offset_max_force(self, rel_time, acceleration):
```

Parameter	Beschreibung
self	Instanz-Referenz
rel_time	Relative Zeit von Max-Force
acceleration	Beschleunigungsvektor

Die Methode `__calculate_offset_max_force` berechnet, wann die grösste Kraft in der Messung auftrat und gibt den Zeitpunkt als Offset zurück.

4.3.10 Methode: `__calculate_custom_offset_force`

```
def __calculate_custom_offset_force(self, custom_offset, acceleration):
```

Parameter	Beschreibung
<code>self</code>	Instanz-Referenz
<code>custom_offset</code>	Relative Zeit, custom Offset
<code>acceleration</code>	Beschleunigungsvektor

Die Methode `__calculate_custom_offset_force` berechnet, wann die Kraft zu einem beliebigen Offset in der Messung und gibt diese Kraft zurück.

4.3.11 Methode: `__calculate_angle`

```
def __calculate_angle(self, index, predicted_impact_time, rel_time, rx, ry):
```

Parameter	Beschreibung
<code>self</code>	Instanz-Referenz
<code>index</code>	Index der Maximalen Kraft im rx und ry Array
<code>predicted_impact_time</code>	Die Berechnete Impact Zeit der Max-Force
<code>rel_time</code>	Die Relative Zeit des Einschlags
<code>rx</code>	Array der X-Achsen Beschleunigungen
<code>ry</code>	Array der Y-Achsen Beschleunigungen

Die Methode `__calculate_angle` berechnet mittels euklidischer Distanz den Einschlags-Winkel anhand der Kräfte Vektoren.

4.3.12 Methode: `__ringbuffer2array`

```
def __ringbuffer2array(self, ringbuffer):
```

Parameter	Beschreibung
<code>self</code>	Instanz-Referenz
<code>ringbuffer</code>	Input JSON Daten

Da die Daten im JSON als Ringbuffer gespeichert werden, sortier die Methode `__ringbuffer2array` die Werte zuerst chronologisch.

4.3.13 Methode: `__read_json_from_filesystem`

```
def __read_json_from_filesystem(self, path2file):
```

Parameter	Beschreibung
<code>self</code>	Instanz-Referenz
<code>path2file</code>	Input JSON Daten als Pfad oder direkt als String

Die Methode `__read_json_from_filesystem` liest die Input JSON Daten entweder vom Filesystem oder direkt aus einem String.

4.3.14 Methode: `__get_b64payload_from_basejson`

```
def __get_b64payload_from_basejson(self, base_json):
```

Parameter	Beschreibung
<code>self</code>	Instanz-Referenz
<code>base_json</code>	Input JSON Daten

Die Methode `__get_b64payload_from_basejson` extrahiert den Base64 Payload aus dem Input JSON.

4.3.15 Methode: `__encoded_payload_to_list`

```
def __encoded_payload_to_list(self, encodedjsonstring):
```

Parameter	Beschreibung
<code>self</code>	Instanz-Referenz
<code>encodedjsonstring</code>	Input JSON Daten als encodierter String.

Die Methode `__encoded_payload_to_list` konvertiert den encodierten JSON String in eine Python List zur besseren weiterverarbeitung.

4.4 Implementierung im Projekt

Innerhalb von dem Projekt werden wir die Funktion `parse_input_data` wie folgt benutzt:

Die Json Daten werden an die Klasse "data_parse" übergeben, welche danach die Rückgabewerte Kraft, Winkel und Zeit liefert. Diese Informationen werden einem drawer-Objekt übergeben, welches anhand dieser Parameter das Einschlagsbild zeichnet (siehe Kapitel 4.5 & 4.6)

5 Damage drawer

5.1 Funktionsumfang

Mit der Klasse DamageImage (File: `damage_image.py`) werden alle Funktionalitäten im Zusammenhang mit der Bild zusammengebündelt. Die Umfasst als Beispiel das Erkennen der Kulturen von dem Auto, das Zeichnen der Beschädigung wie auch das Beschriften der Milisekunden des Aufpralls, die Crash-ID und weiter Informationen.

5.2 Funktionsdesign

5.2.1 Software Abhängigkeiten

Für die Erkennung der Kulturen wurde die Bildverarbeitungs Library OpenCV (Open Source Computer Vision Library) verwendet. Die Kulturen werden verwendet um den Eintrittspunkt der Beschädigung zu berechnen.

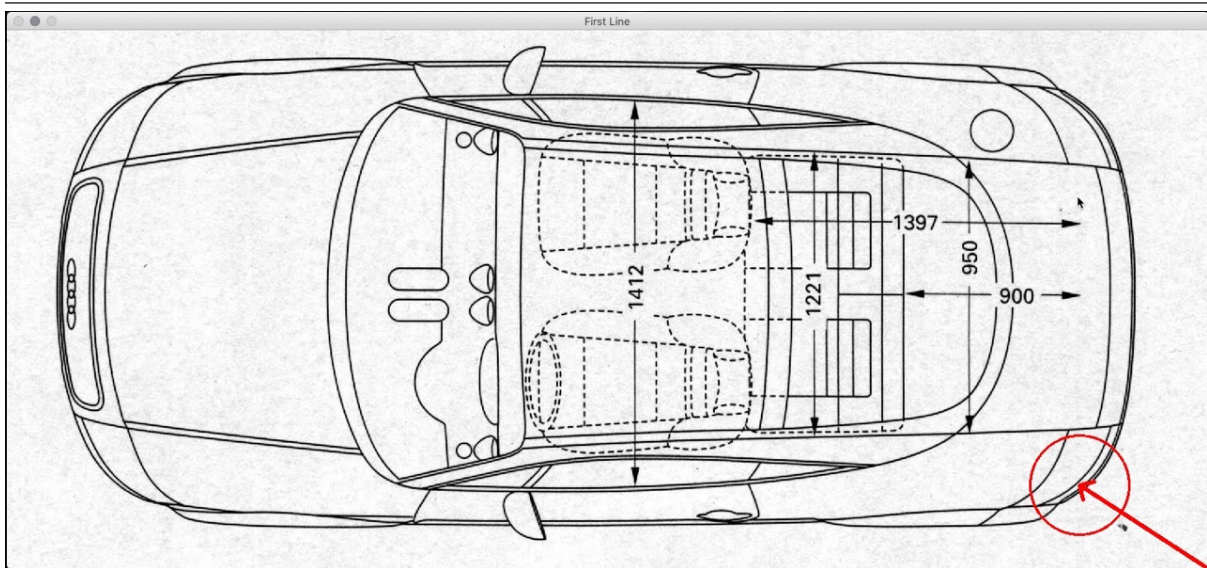
Für die Berechnung innderhalb der Klasse DamageImage wurde auf die bekannte Python Library Numpy (<http://www.numpy.org>) zurückgegriffen.

Die Python Standardbibliothek `os / math / shutil` werden für kleinere Funktionen benötigt.

5.2.2 Prozess des Funktionsdesigns

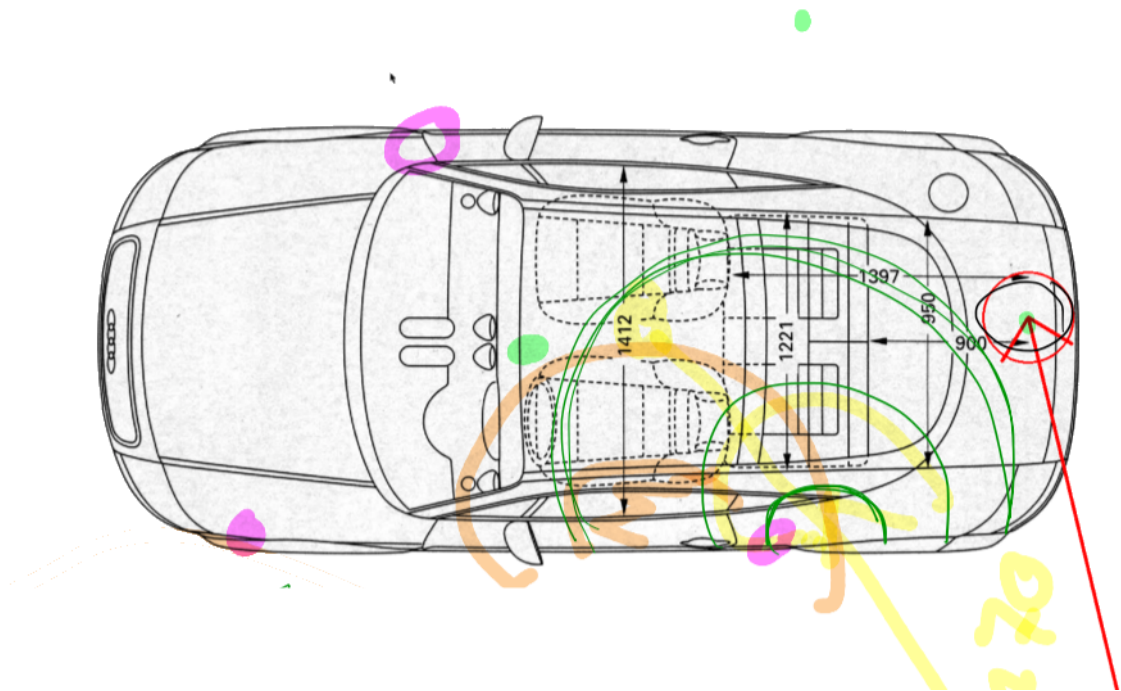
Zu Beginn wurde mittels OpenCV die Datei eingelesen und hardcoded ein Kreis und ein Pfeil gezeichnet. Mit dieser Version haben wir dann im Team das Zieldesign der Bilddatei mittels iPad und Pen gezeichnet.

Erste Version Damage Image



Skizze Damage drawer

Rendered crash image after 6110[ms]

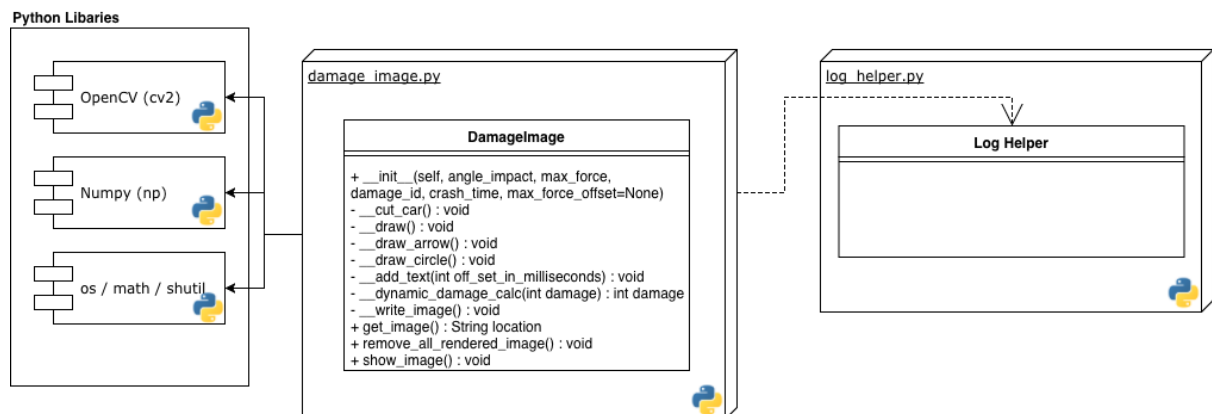


5.3 Realisation / Umsetzung

Klassendiagramm und Beschreibung der Funktionen der einzelnen Methoden.

5.4 Klasse DamageImage

Teilfunktion: Damage Image



5.4.1 Methode: Init

```
def __init__(self, angle_impact, max_force, damage_id, crash_time, max_force_offset=None):
```

Die Init Methode wird aufgerufen bei der Instanzierung von einem Objekt. Als Parameter werden folgende Informationen benötigt:

Parameter	Beschreibung
self	Instanz-Referenz
angle_impact	Winkel in Grad. Mit diesem Winkel wird der Aufprall der Beschädigung gezeichnet.
max_force	Numerischer Wert mit dem die Grösse der Beschädigung am Auto berechnet wird.
damage_id	String. Eindeutiger Crash-Report ID
crash_time	String. Uhrzeit für die Beschriftung auf dem Bild
max_force_offset	Numerischer Wert. Zeitpunkt, nach wie vielen Millisekunden die Beschädigung berechnet wurde (Default=None)

Mit den Methoden Parameter werden die jeweiligen lokalen / privaten Methodenvariablen initialisiert und den Wert zugewiesen. Weiter wird der Pfad wo die Bilddatei gespeichert wird aufgrund der damage_id und der allfälligen max_force_offset. Die Dateien werden in einem speziell definierten Ort (images_rendered/) gespeichert. Dies hat den Grund, dass wenn die Bilddatei von den gleichen Crash-Report und dem gleichen max_force_offset nicht nochmals neu erstellen und schreiben muss, sondern direkt zurückgeben kann.

5.4.2 Methode: Kulturen Auto

```
def __cut_car(self):
```

Parameter	Beschreibung
self	Instanz-Referenz

Die Methode `__cut_car` findet mittels OpenCV die Kulturen (Randkulturen) von dem Auto-Bild. Diese Kulturen werden als Pixel-Matrix abgespeichert und später für den Eintrittspunkt der Beschädigung benötigt. Dazu wird das Auto-Bild in Schwarz/Weiss konvertiert und den Threshold für die Linien gesetzt. Mittels der OpenCV Funktion `cv2.findContours` können die äussersten, geschlossene Linien abgefragt werden.

5.4.3 Methode: Zeichnen

```
def __draw(self):
```

Paramenter	Beschreibung
self	Instanz-Referenz

Die generelle Zeichnungsmethode `__draw` ist als Wrapper Methode zu verstehen. Wenn gegebenenfalls noch weitere Beschriftungen auf die Bilddatei geschrieben werden soll, kann dies hier eingefügt werden. Hier passiert auch die Entscheidung ob der Text für den Offset der Millisekunden angezeigt wird oder nicht.

5.4.4 Methode: Zeichnen - Pfeil

```
def __draw_arrow(self):
```

Paramenter	Beschreibung
self	Instanz-Referenz

Hier wird der Pfeil für die Bilddatei gezeichnet. Etwas unschön ist hier, dass auch noch der Nullpunkt des Koordinatensystem in dieser Methode gezeichnet wird. Eine entsprechendes `# TODO:` ist hier vermerkt.

5.4.5 Methode: Zeichnen - Kreis

```
def __draw_circle(self):
```

Paramenter	Beschreibung
self	Instanz-Referenz

Der Kreis für die für die Grösse der Beschädigung wird hier auf die Bilddatei gezeichnet. Die Grösse von dem Kreis (Radius) wird mit Hilfe der Funktion `__dynamic_damage_calc` berechnet.

5.4.6 Methode: Text auf das Bild

```
def __add_text(self, off_set_in_milliseconds):
```

Paramenter	Beschreibung
self	Instanz-Referenz

Parameter	Beschreibung
off_set_in_milliseconds	Numerischer Wert. Zeitpunkt, nach wie vielen Millisekunden die Beschädigung berechnet wurde

Auf der resultierende Bilddatei werden folgende Informationen dargestellt:

- Time-Offset von der maximalen Beschädigung
 - Text: Rendered crash image after " off_set_in_milliseconds + "[ms]"
- Eindeutige Bilddatei Beschriftung mit der entsprechender Uhrzeit aus dem Crashreport
 - Text: crash identifier = " + self.damage_id + " - damage time = " crash_time

5.4.7 Methode: Berechnung Beschädigung

```
def __dynamic_damage_calc(self, damage):
```

Parameter	Beschreibung
self	Instanz-Referenz
damage	Numerischer Wert mit dem die Grösse der Beschädigung am Auto berechnet wird.

Diese Hilfsmethode berechnet aufgrund einem numerischen Wert die grösse der Beschädigung. Die maximale Beschädigung von 15 und die minimale Beschädigung von 2 wurden aus den Daten ermittelt.

5.4.8 Methode: Schreiben der Bilddatei

```
def __write_image(self):
```

Parameter	Beschreibung
self	Instanz-Referenz

Hier wird die PNG-Bilddatei auf den lokalen Storage von dem Server heruntergeschrieben.

5.4.9 Methode: Pfad der geschriebene Datei

```
def get_image(self):
```

Parameter	Beschreibung
self	Instanz-Referenz

Diese Public Methode wird von Server verwendet um die fertige Bilddatei zu erhalten. Innerhalb dieser Methode werden die Zeichnungsmethoden self.__draw() und die self.__write_image() ausgeführt.

Als Rückgabewert erhält der Server den Pfad der Datei, welche auf dem Server geschrieben wurde.

5.4.10 Methode: Löschen von allen gerendert Dateien

```
def remove_all_rendered_image(self):
```

Parameter	Beschreibung
self	Instanz-Referenz

Diese Housekeeping Methode dient dazu, alle gerenderten Bilddateien auf dem Server zu löschen. Die Methode kann vor dem Start des Server wie auch nach dem Testing ausgeführt werden.

5.4.11 Methode: Anzeige der Datei auf dem Bildschirm

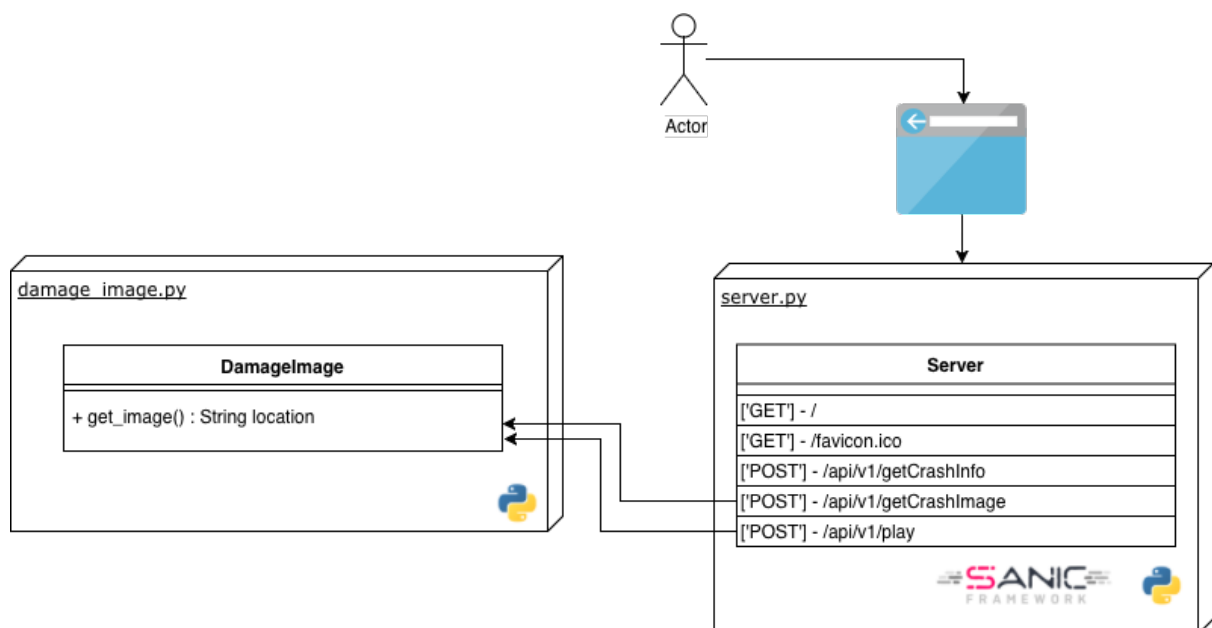
```
def show_image(self):
```

Parameter	Beschreibung
self	Instanz-Referenz

Die show_image dient für eine rasche Entwicklung ohne Server. Sie führt die gleichen Methoden aus, wie die Methode get_image mit dem Unterschied, dass die Bilddatei nicht an den Server zurückgegeben wird sondern mittels OpenCV an dem Display angezeigt wird. So kann Abhängigkeit vom Server neue Funktionen rasche getestet werden.

5.5 Implementierung im Projekt

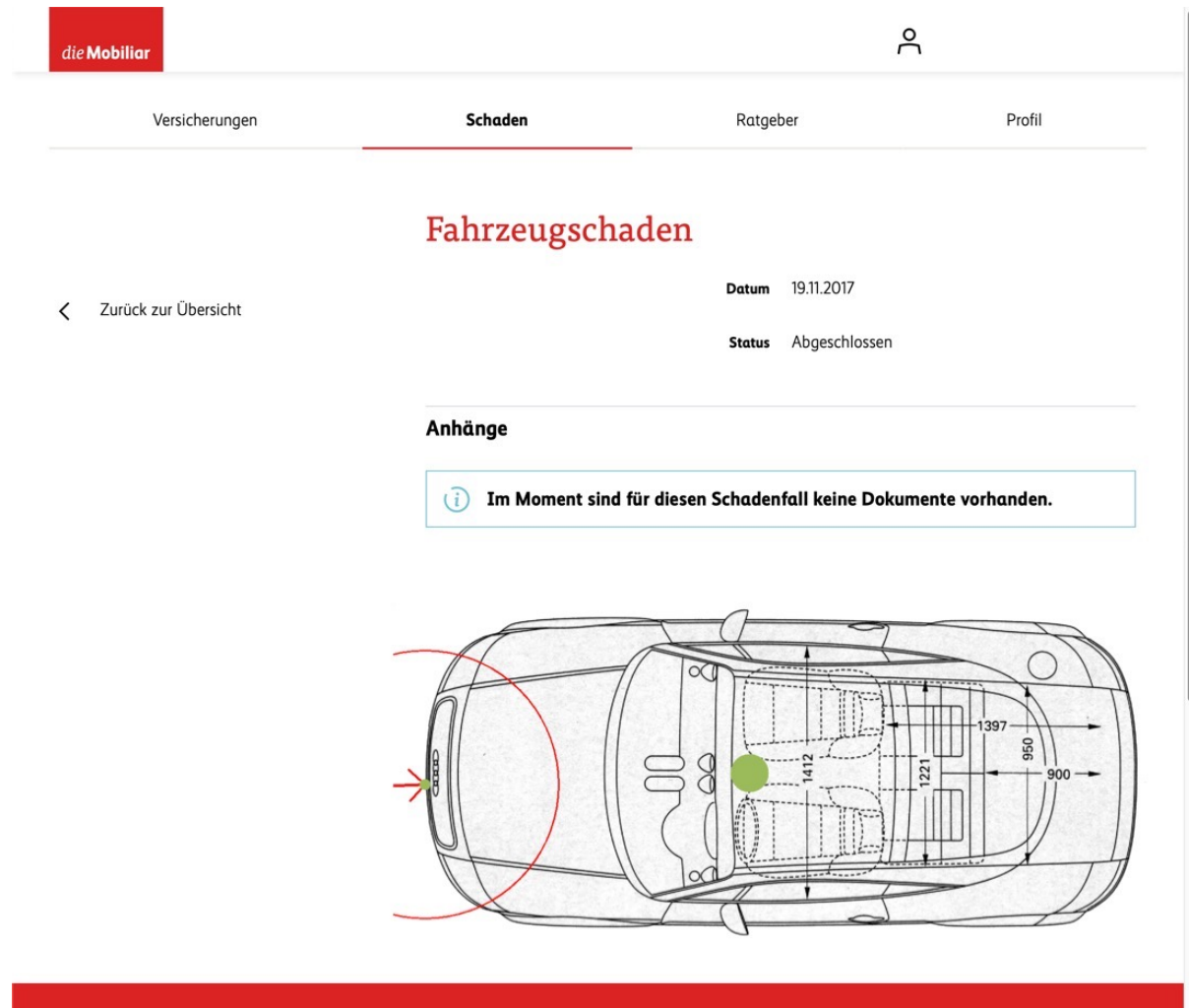
Innerhalb von dem Projekt werden wir die Funktion get_image wie folgt benutzt:



Innerhalb vom server.py wird ein Damage drawer Objekt erstellt und mittels der Funktion get_image die fertig gerenderte Bilddatei zurückgegeben und auf der Webseite dargestellt.

5.6 Mögliche Darstellung der Datei in einem Portal

Ein möglicher Einsatzbereich von unserem Projekt könnte ein Portal von einer Versicherung sein. Hier würde bei Autounfällen der Ort von dem Schaden wie auch das Ausmass der Beschädigung aufgezeigt werden. So kann ein Mehrwert in Form von mehr Informationen an dem Kunde einer Autoversicherung entstehen.



6 Server / Frontent und Backend

tbd