MANUAL

Mapillary-Based Damage Annotator

COVID-19 GLOBAL RESPONSE

Corina Markodimitraki, Melissa El Hamouch and Georgios Roullis

2021.05.20 – Version 1

With support of

For more information regarding the use of this product, please contact the 510 Data Team by sending an email to support@510.global

Contents

Introduction

Since March 2020, the 510 Data Team of the Netherlands Red Cross (NLRC) has supported the Lebanese Red Cross (LRC) by providing a framework to assess the damage caused by the Beirut blast in August 2020. This manual provides the information that one needs to know when planning on using this framework to assess damage.

Product in a nutshell

The framework presented in this manual consists of programming scripts and an online webpage that support the collection of photographic material and analysis thereof for possible damage detection and assessment.

What is the Mapillary-based damage annotator and what can you do with it?

The 510 Data Team has used the Mapillary-based damage annotator to assess the damage caused by the explosion that hit Beirut, on August 4th 2020, thereby supporting the Lebanese Red Cross. This is done in the following way:

1. By collecting photographs taken of street views after the blast
2. By subsequently analyzing the photos and calculating an overall damage score

To collect photographs with a street view, the 510 Data Team uses an API (Automatic Programming Interface) to download publicly available photos from Mapillary (www.mapillary.com). They also use several Python scripts and the Oxford VGG image annotator (https://www.robots.ox.ac.uk/~vgg/software/via/via_demo.html), a manual classification tool to analyze the data and assess damage seen in the photos. Finally, an overall damage score is calculated. The damage score can be visualized on a map using a GIS (Geographic Information System) program such as QGIS.

Who is Mapillary-based damage annotator for?

The annotator can be used by any National Society (NS) or other non-governmental organisation looking to quickly evaluate the damage caused by a natural or man-made disaster and organize their response. Until now the 510 Data Team has aided the Lebanese Red Cross in the assessment of the damage caused by the Beirut blast on August 4th 2020.

Why use the Mapillary-based damage annotator?

The annotator presented in this manual offers an open source, fast and semi-automated way of assessing damage caused to an area of interest. The annotator can help non-governmental organizations such as a NS, with fast decision making and response to help populations in the severely damaged areas. What’s more, the annotator facilitates the collection and analysis of large amounts of data (if available on Mapillary), and allows the user to repeat this process as often as they desire.

Case study

The Mapillary-based damage annotator was used to support the LRC in assessing the damage caused to the city after the explosion on August 4th 2020. The 510 Data Team was able to support in mapping out the damage done to the city, in order for LRC to focus their efforts with on the ground support where needed. The main advantage of using this tool is that the damage annotation can be done remotely and is not required to be used in the area affected by damage. The street view photographs are collected by the general population or volunteers on the ground and can then be uploaded to the the open-source Mapillary site. From there, remote volunteers can download and analyze the photos to calculate the overall damage done.

For the case of Beirut, a total of 13 volunteers were needed to manually assess the damage shown in over 1,000 images. Each image was checked by 3 volunteers for validation. The assessments were done in a span of 12 hours, thus making the tool a fast and efficient method for damage assessment.

For more information on how data played a role after the Beirut Explosion, please follow the link below:

https://www.510.global/beirut-after-the-explosion-how-technology-is-playing-a-vital-role/

Data responsibility

Datasets

The type of data used are photographs taken at street view level. The geographic location will be identifiable to indicate the area of which the damage is made. Additionally, the person annotating will have to save the annotated photographs in a folder that might, at time, indicate their name.

Data processing

The data is collected from Mapillary, which is an open-source site. Photographs can be downloaded automatically using an Automatic Programming Interface and the annotation is done in a semi-automated way.

Non-discrimination

This tool is non-discriminatory.

Human oversight

The tool is dependent on one or multiple people who will manually assess the damage shown in the images. The calculation of the overall risk depends on the availability of people, which might be a limitation if the number of people needed is not attained. Each image should be annotated by 3 or more people to ensure accuracy.

Risks

The only foreseeable risk is the ability to identify geographic locations with ease, this might cause some issues if the designed maps are used for the wrong reasons.

Requirements

User knowledge

In this section we describe the skills that someone would need to use the annotator presented in this manual. In addition to the skills mentioned below, it is crucial the user has the following skills:

- Analytical thought
- Problem-solving attitude
- Data responsibility (see also section “Data responsibility” of this manual)
- GIS skills

To collect and analyze Twitter data the user will need the following skills:

- **Mapillary**: it is desirable that the user has knowledge and experience with Mapillary.
- **Python**: it is necessary that the user has knowledge and experience with Python to execute the API and run the Python scripts.
- **GEOJSON**: it is necessary that the user has knowledge and experience with geojson files.
- **GIS skills**: in case the user wants to visualize the collected data, they will need to have knowledge of QGIS desktop (or any other GIS program).

Hardware

Below you can find the hardware requirements for the Mapillary-based damage annotator explained in this manual.

To collect and analyze the data:

- **Python** (versions: 2.7.X, 3.6.X):
    - Processor: Intel Atom® processor or Intel® Core™ i3 processor
    - Disk space: 1 GB
    - Operating system: Windows* 7 or later, macOS, or Linux

To visualize the data:

- **QGIS**:
    - Processor: minimum Core i3 2.7Ghz; recommended Core i7 3.5Ghz

- RAM: minimum 2Gb; recommended 8Gb or more
- Disk space: minimum 500 Gb SATA; recommended SSD 128Gb or 500Gb SATA
- Graphic card: minimum 1Gb RAM; recommended 2Gb RAM (NVIDIA Geforce)
- Operating system: Windows 7-10, Mac OSX, Linux, Unix, or Android

Software

Below we list the software required for the Mapillary-based damage annotator explained in this manual.

To collect, analyze and visualize the data

- Internet browser (for computers) such as
    - Mozilla Firefox
      www.mozilla.org/en-US/firefox/new/
    - Google Chrome
      www.google.com/chrome/
- Installed Python
  https://www.python.org/
- (Optional) Installed QGIS
  https://www.qgis.org/en/site/

Time

Here we indicate which parameters influence the duration of usage of the Mapillary-based damage annotator, and state a real-life example of usage by a NS.

The duration of setting up and using the Mapillary-based damage annotator is the total duration of the following steps: downloading the image data, running the scripts, annotating the damage, calculating the damage score and optionally, visualizing it. All this depends on the number of images that need to be annotated and user skills in Python.

For reference, the 510 Data Team of the NLRC spent the following amount of time for each step in the annotation process for the damage assessment of the Beirut blast:

- **To run the script and collect the images**: a total of 1,432 images were collected from Mapillary in a maximum of 10 minutes.

- **To assess the damage by means of manual work**: a total of 13 volunteers were needed to manually assess the damage shown in the 1,432 photographs within a total time of 12 hours. Each image was evaluated by 3 different volunteers.

The product

What does the Mapillary-based damage annotator consist of?

The annotation of damage can be achieved by using all of the following:

- (Optionally) Bitwarden to acquire the access token for Mapillary
- A Mapillary API (called within a Python script)
- Python scripts
- The Oxford VGG image annotator
- (Optionally) QGIS or any other GIS program

Locations to download the product

Here we provide the links and locations to find the tools needed to set up and use the Mapillary-based damage annotator.

- (Optional) Bitwarden to acquire the access token for Mapillary
  https://bitwarden.com/
- Github page containing the Python scripts and information needed to run these
  https://github.com/rodekruis/building-damage-classification-mapillary
- The Oxford VGG image annotator
  https://www.robots.ox.ac.uk/~vgg/software/via/via_demo.html
- (Optional) GIS visualization tool such QGIS
  www.qgis.org

Advantages & Limitations

There are several advantages and limitations linked to using the Mapillary-based damage annotator. Here we briefly discuss these.

- The user is depended on the data that is available on Mapillary. This depends on internet access, available means to capture photos and a familiarity of the country's population with Mapillary. This could translate to a considerable number of available images (see Beirut blast example where 1.432 images were collected), but also to a bottleneck in the pipeline if there are no images available. However, if needed, on-ground volunteers can take photographs and upload them to Mapillary.

- The exact location of the potential damage is available due the image being linked to a map through Mapillary. This offers the user very precise information which is helpful when addressing the needs after the caused damage.
- The pipeline described in this manual requires expertise in Python to setup, which could be a limitation if there is no expert present in the organization aiming to use the annotator, although less skill is needed to run the pipeline once it is set up.
- The organization using the annotator is depended on the presence of people to manually assess the damage in the images. If there is a limited amount of people available to do the assessment, it could delay the process of calculating the overall risk score.

Setting up the Mapillary-based damage annotator: a step-by-step guide

1. Register organization at Bitwarden (optional)

Bitwarden is a free and open-source password management service that stores sensitive information such as website credentials in an encrypted vault. We recommend using Bitwarden to store the login information of any website your organization uses. If your organization is not yet registered at Bitwarden (www.bitwarden.com), registration is recommended to secure login information of the organization to websites. Using Bitwarden however, is not a prerequisite to using the Mapillary-based damage annotator.

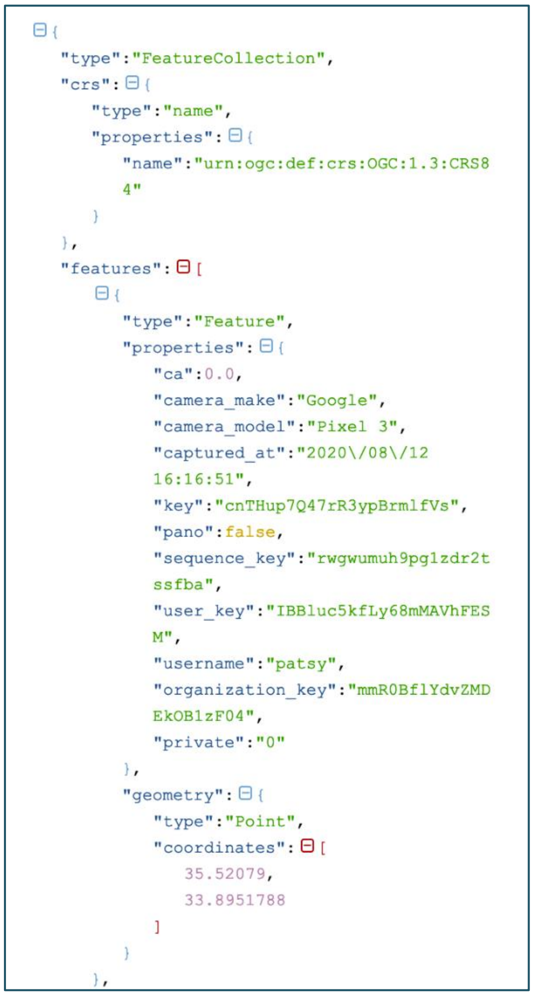
2. Collect relevant images from Mapillary

For this step, the Python script `get_mapillary_images.py` is needed. Here, Mapillary (www.mapillary.org) is used as the primary source for geotagged photos. Mapillary offers open source, accessible image data submitted by the platforms' users. By using the `get_mapillary_images.py` script:

a. Acquire an access token for Mapillary from Bitwarden.

b. Define the parameters which the images have to fulfill for collection. The script defines the date (submitted images from this day onwards) and the location (by means of a bounding box) as well as the format (geojson, see **Figure 1**) and download folder of the outputted images.

c. Run the script to collect the images in the specified folder.



```
⊟{
   "type":"FeatureCollection",
   "crs":⊟{
      "type":"name",
      "properties":⊟{
         "name":"urn:ogc:def:crs:OGC:1.3:CRS8
         4"
      }
   },
   "features":⊟[
      ⊟{
         "type":"Feature",
         "properties":⊟{
            "ca":0.0,
            "camera_make":"Google",
            "camera_model":"Pixel 3",
            "captured_at":"2020\/08\/12
            16:16:51",
            "key":"cnTHup7Q47rR3ypBrmlfVs",
            "pano":false,
            "sequence_key":"rwgwumuh9pg1zdr2t
            ssfba",
            "user_key":"IBBluc5kfLy68mMAVhFES
            M",
            "username":"patsy",
            "organization_key":"mmR0BflYdvZMD
            EkOB1zF04",
            "private":"0"
         },
         "geometry":⊟{
            "type":"Point",
            "coordinates":⊟[
               35.52079,
               33.8951788
            ]
         }
      },
```

Figure 1 • Part of the collected image data within the geojson file. The file is opened in http://json.parser.online.fr/

3. **Define the labels and features for the annotation process**

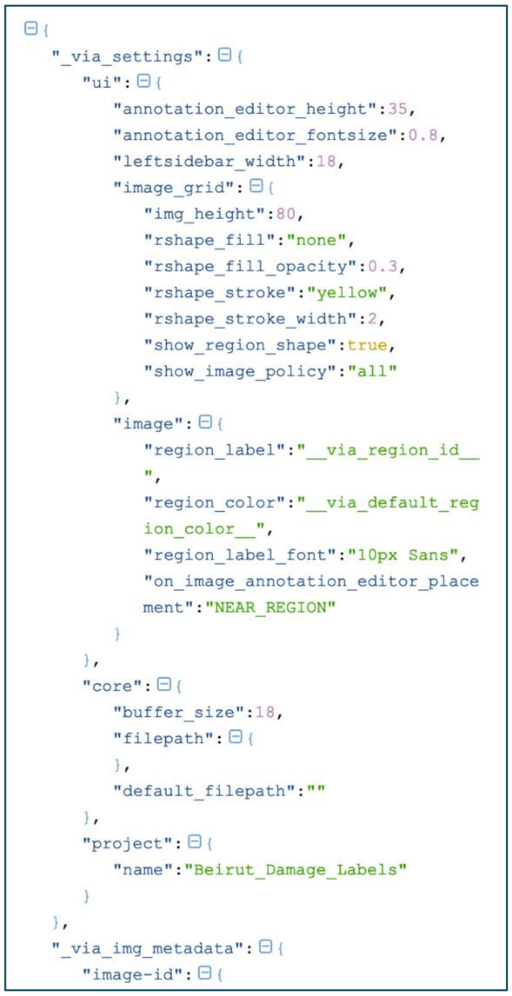
   For this step, the Python script `labelling_project_config.py` is needed. This script is the configuration file used to define the labels for the features that are going to be used for the annotation process.

   a. Define the features for the annotation. Features are the elements one should assess for possible damage such as windows, balconies etc. For example, to assess if windows are broken, the label "`window_damage`" should be defined.

   b. Define the importance of possible damage of the chosen features and make the distinction between light, medium and severe damage. For example, light damage is indicated with debris (`light_features = ['debris']`), while medium damage is indicated with damage to windows (`medium_features = ['balcony_damage']`).

4. **Split the images into batches for the annotation**

   For this step, the Python script `generate_batches.py` is needed. This script splits the acquired images of the geojson file into batches and prepares one "project" per batch in json format, that will be used later on for the Oxford VGG image annotator. An example of a project template in json format, `labelling_project_template.json` (found on the Github page of the annotator), is shown in **figure 2**.



```
⊟{
   "_via_settings":⊟{
      "ui":⊟{
         "annotation_editor_height":35,
         "annotation_editor_fontsize":0.8,
         "leftsidebar_width":18,
         "image_grid":⊟{
            "img_height":80,
            "rshape_fill":"none",
            "rshape_fill_opacity":0.3,
            "rshape_stroke":"yellow",
            "rshape_stroke_width":2,
            "show_region_shape":true,
            "show_image_policy":"all"
         },
         "image":⊟{
            "region_label":"__via_region_id__
            ",
            "region_color":"__via_default_reg
            ion_color__",
            "region_label_font":"10px Sans",
            "on_image_annotation_editor_place
            ment":"NEAR_REGION"
         }
      },
      "core":⊟{
         "buffer_size":18,
         "filepath":⊟{
         },
         "default_filepath":""
      },
      "project":⊟{
         "name":"Beirut_Damage_Labels"
      }
   },
   "_via_img_metadata":⊟{
      "image-id":⊟{
```

Figure 2 • Part of the project template json file, opened in http://json.parser.online.fr/

5. **Manually annotate the images**

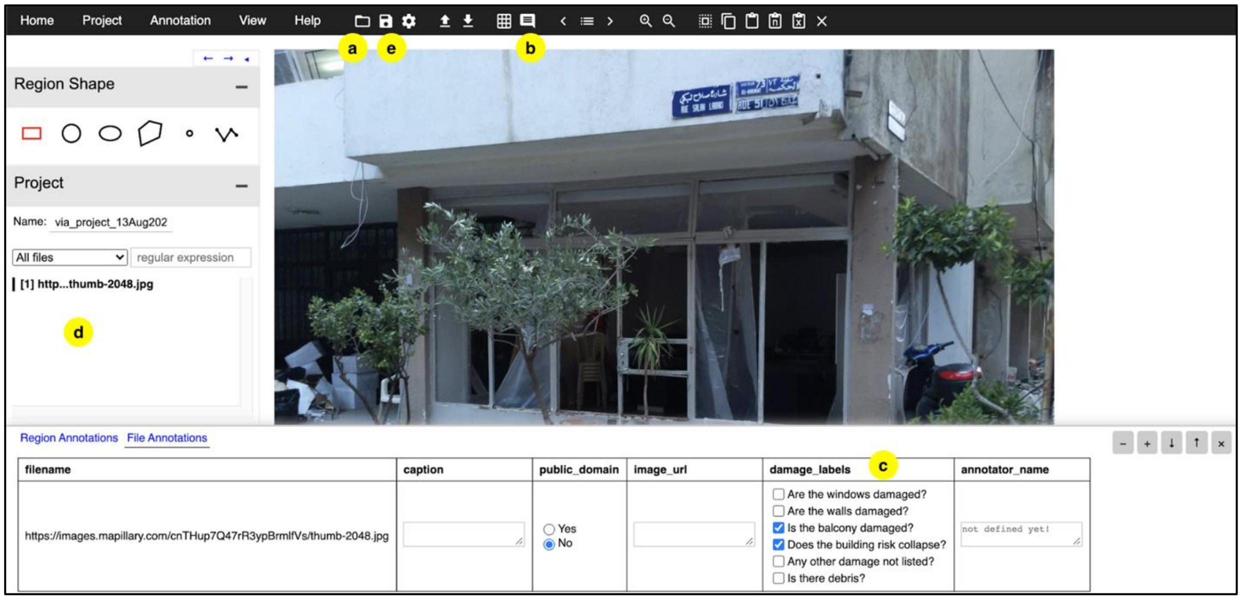
   For this step, the Oxford VGG image annotator is needed (see section "The product" of this manual) as well as people to assess the images. As a standard, we recommend at least 3 people assess the image independently, to avoid mislabeling bias.

   a. Click "Open Project" (folder icon) and select the json file containing the desired batch for analysis (**figure 3a**).

b. Click “Toggle Annotation Editor” (speech bubble icon) to show annotation tab (bottom) and click “File Annotations” in the annotation tab (**figure 3b)**.

c. For each image, mark damage: click/select if there is damage, do NOT click if there’s NO damage (**figure 3c)**. If there is uncertainty about the origin of the damage, (e.g., is the damage a result of the disaster or was it already there?), copy-paste the image_url (found under filename) in the browser and see how distant the building from disaster location. It’s recommended to keep the expected damage radius into account.

d. Scroll through the images in the project tab (left) (**figure 3d)**.



filename	caption	public_domain	image_url	damage_labels	annotator_name
https://images.mapillary.com/cnTHup7Q47rR3ypBrmlfVs/thumb-2048.jpg		Yes [✓ No]		☐ Are the windows damaged? ☐ Are the walls damaged? ☑ Is the balcony damaged? ☑ Does the building risk collapse? ☐ Any other damage not listed? ☐ Is there debris?	not defined yet!

Figure 3 • The Oxford VGG image annotator interface. The shown image is an example image used for the damage assessment for the Beirut blast. Yellow circles with letters in bold refer to the sub-steps of step 5.

e. Save the project (disk icon) at regular intervals (every 20 images) (**figure 3e)**.

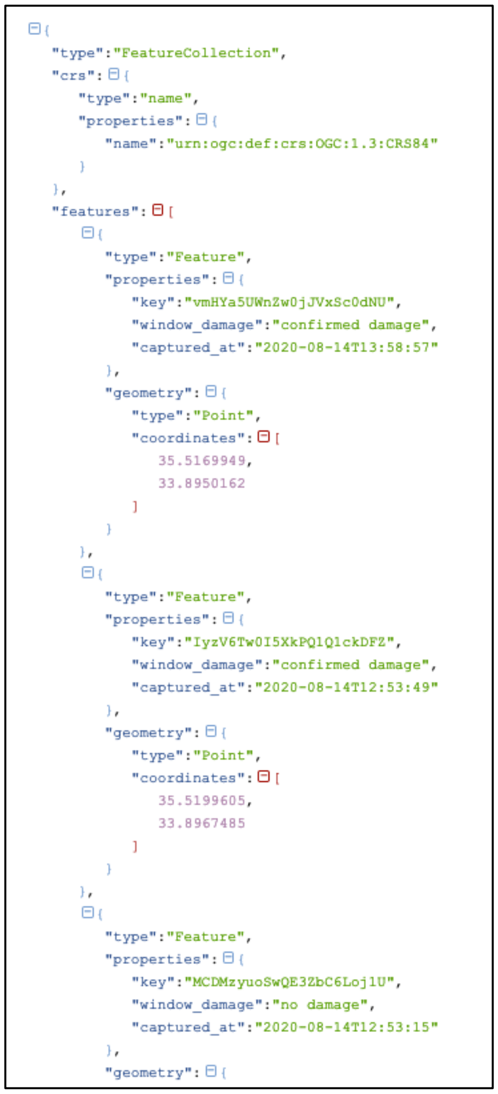
f. Label the batches, get all submissions and save them as `results_batch_<batch number>_<annotator name>.json` (example: `results_batch_12_bob.json`).

6. Merge the submissions and calculate damage score

For this step, the Python script `merge_results.py` is needed. This script fuses all the submissions by all annotators (people) together and:

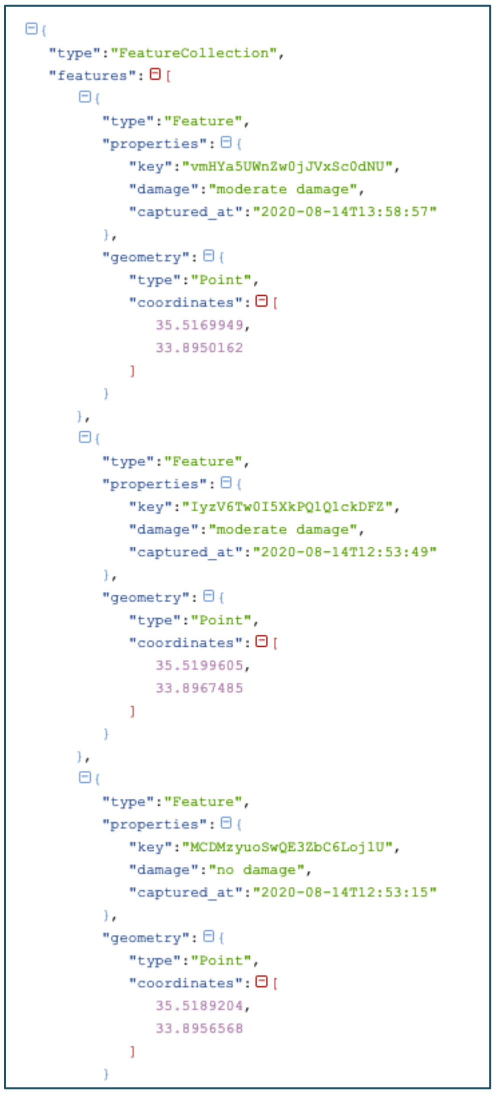
a. saves the information gathered per feature into one geojson vector file which also contains the geographical information of the assessed images (e.g., all information regarding windows is merged, see **figure 4**) and the feature-specific score is calculated per image (for more info on how the score is calculated, please refer to the Github page, see section "The product" of this manual).

b. saves the information gathered of all features into one geojson vector file which also contains the geographical information of the assessed images (e.g., all information regarding windows, balconies, debris is merged, see **figure 5**) and the overall damage score is calculated per image (for more info on how the score is calculated, please refer to the Github page, see section "The product" of this manual).



```
{
   "type":"FeatureCollection",
   "crs":{
      "type":"name",
      "properties":{
         "name":"urn:ogc:def:crs:OGC:1.3:CRS84"
      }
   },
   "features":[
      {
         "type":"Feature",
         "properties":{
            "key":"vmHYa5UWnZw0jJVxSc0dNU",
            "window_damage":"confirmed damage",
            "captured_at":"2020-08-14T13:58:57"
         },
         "geometry":{
            "type":"Point",
            "coordinates":[
               35.5169949,
               33.8950162
            ]
         }
      },
      {
         "type":"Feature",
         "properties":{
            "key":"IyzV6Tw0I5XkPQ1Q1ckDFZ",
            "window_damage":"confirmed damage",
            "captured_at":"2020-08-14T12:53:49"
         },
         "geometry":{
            "type":"Point",
            "coordinates":[
               35.5199605,
               33.8967485
            ]
         }
      },
      {
         "type":"Feature",
         "properties":{
            "key":"MCDMzyuoSwQE3ZbC6Loj1U",
            "window_damage":"no damage",
            "captured_at":"2020-08-14T12:53:15"
         },
         "geometry":{
```

Figure 4 • Part of the geojson file containing information for the feature "windows". The file is opened in http://json.parser.online.fr/



```
{
   "type":"FeatureCollection",
   "features":[
      {
         "type":"Feature",
         "properties":{
            "key":"vmHYa5UWnZw0jJVxSc0dNU",
            "damage":"moderate damage",
            "captured_at":"2020-08-14T13:58:57"
         },
         "geometry":{
            "type":"Point",
            "coordinates":[
               35.5169949,
               33.8950162
            ]
         }
      },
      {
         "type":"Feature",
         "properties":{
            "key":"IyzV6Tw0I5XkPQ1Q1ckDFZ",
            "damage":"moderate damage",
            "captured_at":"2020-08-14T12:53:49"
         },
         "geometry":{
            "type":"Point",
            "coordinates":[
               35.5199605,
               33.8967485
            ]
         }
      },
      {
         "type":"Feature",
         "properties":{
            "key":"MCDMzyuoSwQE3ZbC6Loj1U",
            "damage":"no damage",
            "captured_at":"2020-08-14T12:53:15"
         },
         "geometry":{
            "type":"Point",
            "coordinates":[
               35.5189204,
               33.8956568
            ]
         }
```

Figure 5 • Part of the geojson file containing information for all features. The file is opened in http://json.parser.online.fr/

7. Visualize the damage score on a map (optional)

For this step a Geographic Information System (GIS) program such as QGIS is needed. The damage score per feature, or the overall damage score can be visualized in a number of ways, but a guide to visualization falls outside the scope of this manual. In the example shown in **figure 6**, the damage score for the specific feature "balconies" has been visualized with a color-coded dot: red indicating confirmed damage, yellow indicating possible damage and green indicating no damage. In a similar manner, the overall damage score for all features is shown in **figure 7**. The color code is similar to that of figure 6.

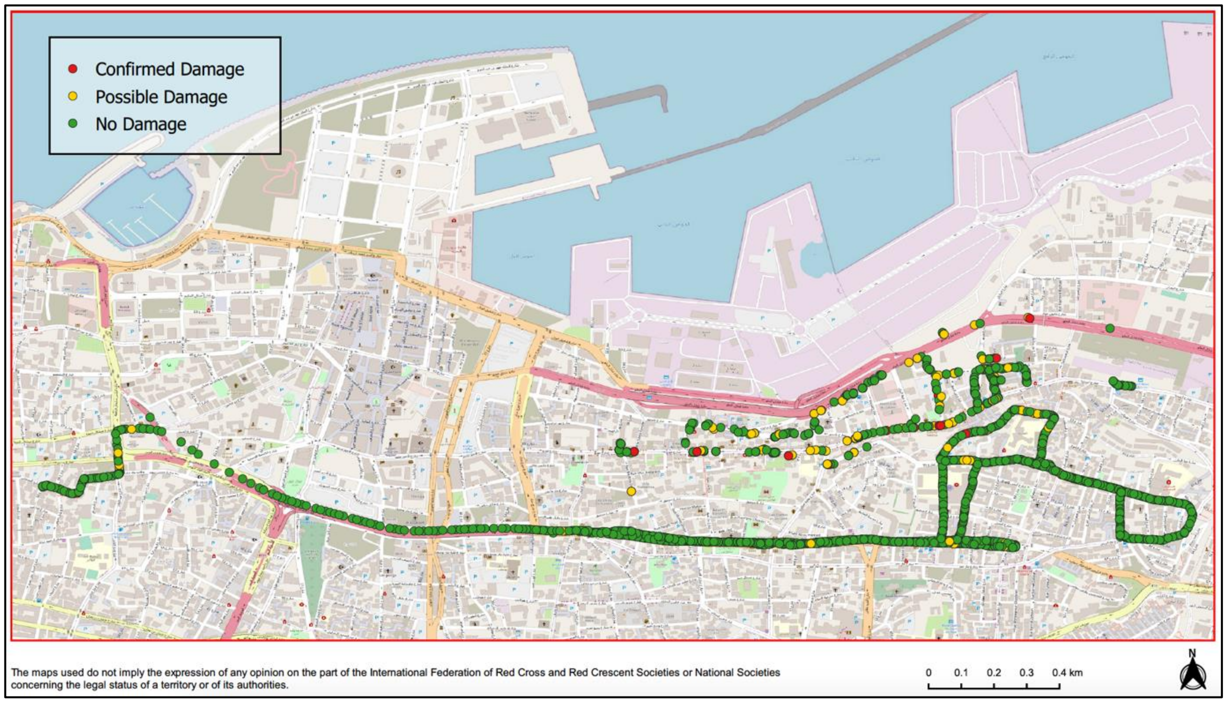

Confirmed Damage
Possible Damage
No Damage
The maps used do not imply the expression of any opinion on the part of the International Federation of Red Cross and Red Crescent Societies or National Societies concerning the legal status of a territory or of its authorities.
0 0.1 0.2 0.3 0.4 km
N

Figure 6 • Example of the feature-specific damage score visualization, in this case balconies. Each circle represents the damage score calculated for that particular image. The damage score was calculated for balconies. Visualization in QGIS.

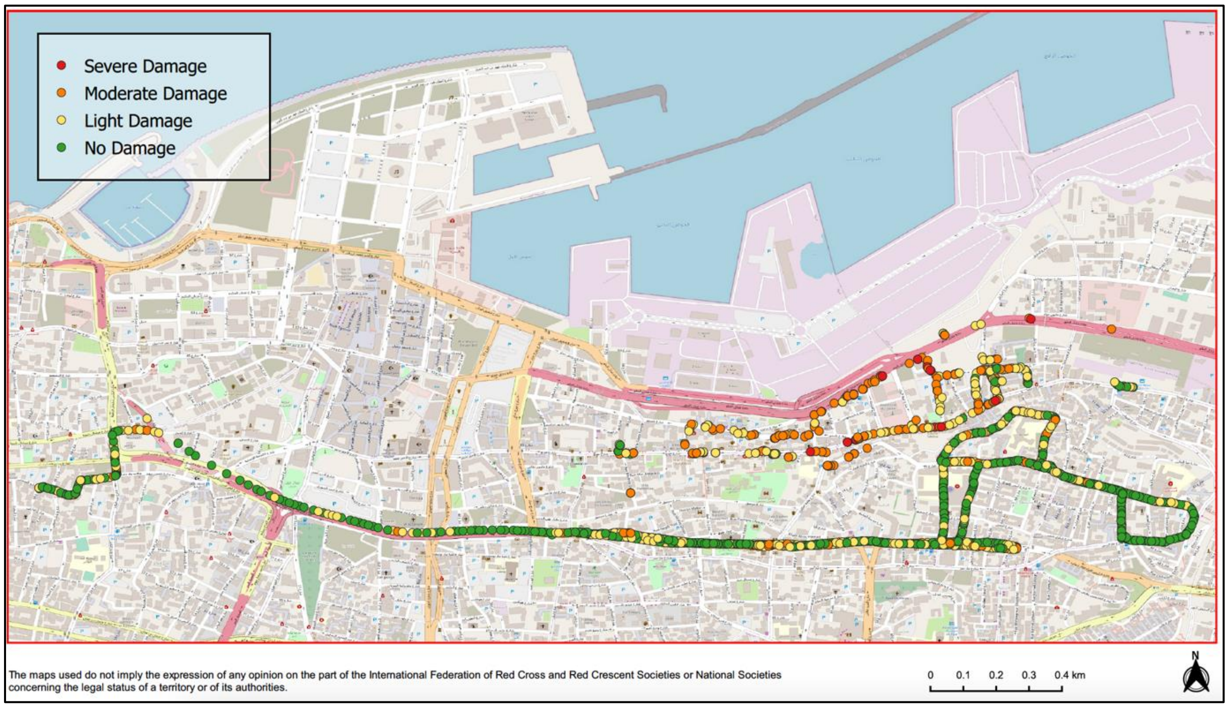

Severe Damage
Moderate Damage
Light Damage
No Damage
The maps used do not imply the expression of any opinion on the part of the International Federation of Red Cross and Red Crescent Societies or National Societies concerning the legal status of a territory or of its authorities.
0 0.1 0.2 0.3 0.4 km
N

Figure 7 • Example of the overall damage score visualization. Each circle represents the damage score calculated for that particular image. Visualization in QGIS.

Abbreviations

NLRC	Netherlands Red Cross
LRC	Lebanese Red Cross
SARS-CoV-2	Severe Acute Respiratory Syndrome coronavirus 2
COVID-19	The disease caused by the SARS-CoV-2 virus
NS	National Society/Societies
API	Application Programming Interface
GIS	Geographic Information System

Resources

- Online program that reads json and geojson files
  http://json.parser.online.fr/
- The Mapillary-based damage annotator Github page
  https://github.com/rodekruis/building-damage-classification-mapillary
- The instructions given to the annotators
  https://drive.google.com/file/d/1nbwxGmjRB_JPdbs3Jp4H7XCoi_hz64xv/view