

Development of Automated Roadside Video Surveys for Detecting and Monitoring Coconut Rhinoceros Beetle Damage

Aubrey Moore

2021-04-29

1

¹The most recent version of this document can be downloaded from
<https://github.com/aubreymoore/roadside/blob/master/docs/roadside/roadside.pdf>.

Contents

1	Background	5
2	Recording Roadside Videos	5
2.1	Mounting a Smart Phone on a Vehicle	5
2.1.1	Setting the Direction of View Angle	6
2.2	Parameter Choices	7
2.2.1	Camera and Lens	7
2.2.2	Resolution	7
2.2.3	Frames per Second	7
2.3	Using the Open Camera App	7
2.4	Using the GPSLogger App	11
2.5	Synchronizing Smart Phone System Time with GPS Time	11
3	Data processing	11
3.1	Georeferencing Video Frames	13
3.2	Mapping	13
4	Detecting CRB Damage in Video Frames	13
5	Creating a CRB Damage Survey Database	13
6	Visualizing Survey Results on a Map	16
7	First Roadside Video Survey of CRB Damage on Guam	19
7.1	Methods	19
7.2	Results	19
8	Extra	20
9	Test Data	21
10	References	21

Listings

1	process-data.py	11
2	create_tables.sql	14
3	create_views.sql	15
4	create_grid.sql	15
5	make-crb-damage-map.py	16

List of Figures

1	Smart phone mount.	6
2	Setting camera angles using Clinometer app.	7
3	Video settings (1/3).	8
4	Video settings (2/3).	9
5	Video settings (3/3).	10
6	Database diagram.	14
7	<code>qgis2web</code> parameters Screenshot 1.	18
8	<code>qgis2web</code> parameters. Screenshot 2.	19
9	Screenshot of an interactive web map of results from the Guam01 roadside video survey of CRB damage on Guam (Moore 2020b) URL: https://aubreymoore.github.io/Guam-CRB-damage-map-2020-10 .	20

1 Background

To see how coconut rhinoceros beetle damage surveys are currently done, please read Jackson 2019 and Vaqalo et al. 2017.

2 Recording Roadside Videos

2.1 Mounting a Smart Phone on a Vehicle

Preliminary work show that mounting the smart phone externally produces much better results than mounting the phone internally as a dash cam. This eliminates problems cause by dirty windshields and internal reflections.

A smart phone can be mounted externally using a ball and socket anchored using a strong magnet (1).

Optimal placement of the smart phone camera appears to be above the right-hand corner of the windshield (passenger side in the US).



Xuma Smartphone Mount

B&H #XUMTA300B • MFR #MTA-300B

Eligible for Free Expedited Shipping on orders over \$49

Accessories



RigWheels RMS1 RigMount

B&H #RIRMS1 • MFR #RMS1

Eligible for Free Expedited Shipping on orders over \$49

Accessories



JOBY BallHead 3K

B&H #JOB01513 • MFR #JB01513

Free Shipping for this Item

Eligible for Free Expedited Shipping on orders over \$49

Accessories

Figure 1: Smart phone mount.

2.1.1 Setting the Direction of View Angle

The direction of view angle has two components, a horizontal angle with respect to direction of travel and a vertical angle. The horizontal angle is set using the scale at the base of the ball joint. The vertical angle is set using a free Android app called Clinometer (<https://play.google.com/store/apps/details?id=net.androgames.clinometer>) (Fig. 2).

Optimal angles for direction of view appear to be 45 degrees to the right of direction off travel and 15 degrees above horizontal.



Figure 2: Setting camera angles using Clinometer app.

2.2 Parameter Choices

2.2.1 Camera and Lens

We are currently using a Samsung Galaxy S10 smart phone. This phone is equipped with four cameras including a 16MP ultra-wide-angle camera (123° field of view) which seems to be a good choice for this application.

2.2.2 Resolution

Maximum resolution for videos recorded with the ultrawide angle camera is 4K 3840x2160 (16:9, 8.29MP). Initial recordings were made using this resolution, but this can probably be reduced without significant loss of precision.

2.2.3 Frames per Second

Standard frame rate is 30 fps. Initial recordings were made using this rate, but this can probably be reduced without significant loss of precision.

2.3 Using the Open Camera App

Open Camera (<https://opencamera.org.uk/>) is a FOSS app for Android smart phones which enables much better control of hardware features than the default Camera app provided with Samsung phones.

Open Camera offers a plethora of settings which can be saved in a configuration file for later use. For screenshots of *Video settings* see figures 3, 4, and 5.

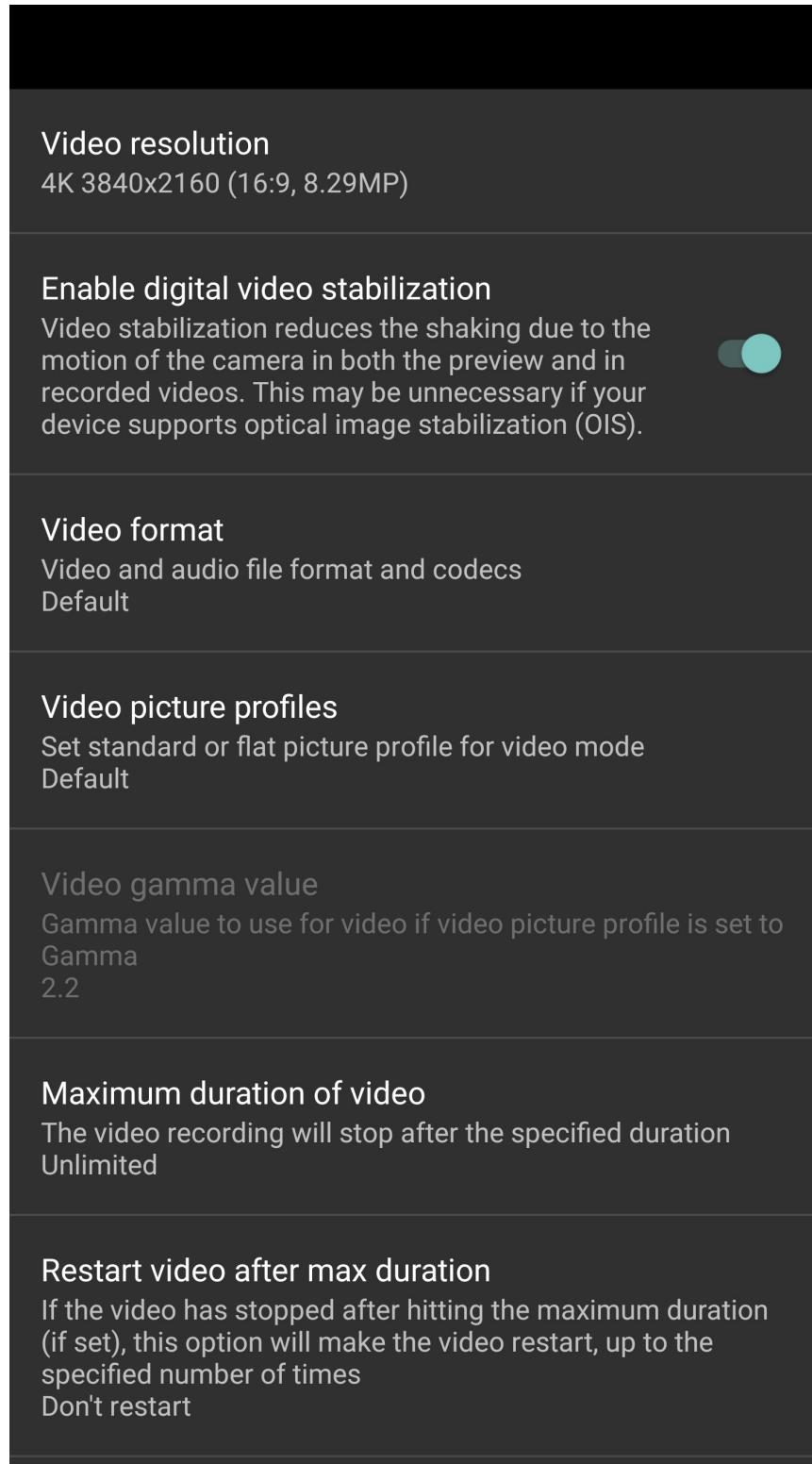


Figure 3: Video settings (1/3).

Maximum file size of video

The video recording will stop and/or restart (see option below) when the (approximate) maximum file size is reached. Note that many Android devices have a maximum file size for videos (typically around 2GB or 4GB), but this option allows setting a specific value. Note that this option can't be used to increase a device's built-in maximum.
100MB

Restart on maximum file size

Whether to automatically restart when the maximum file size is reached (whether the device default max file size, or user specified)



Record audio

Record audio when recording video



Audio source

Microphone to use for recording audio. Note that the exact behaviour of the options depends on how the option is implemented on your device.

Camcorder

Audio channels

Specify mono or stereo for recording audio (stereo only supported on some devices)

Figure 4: Video settings (2/3).

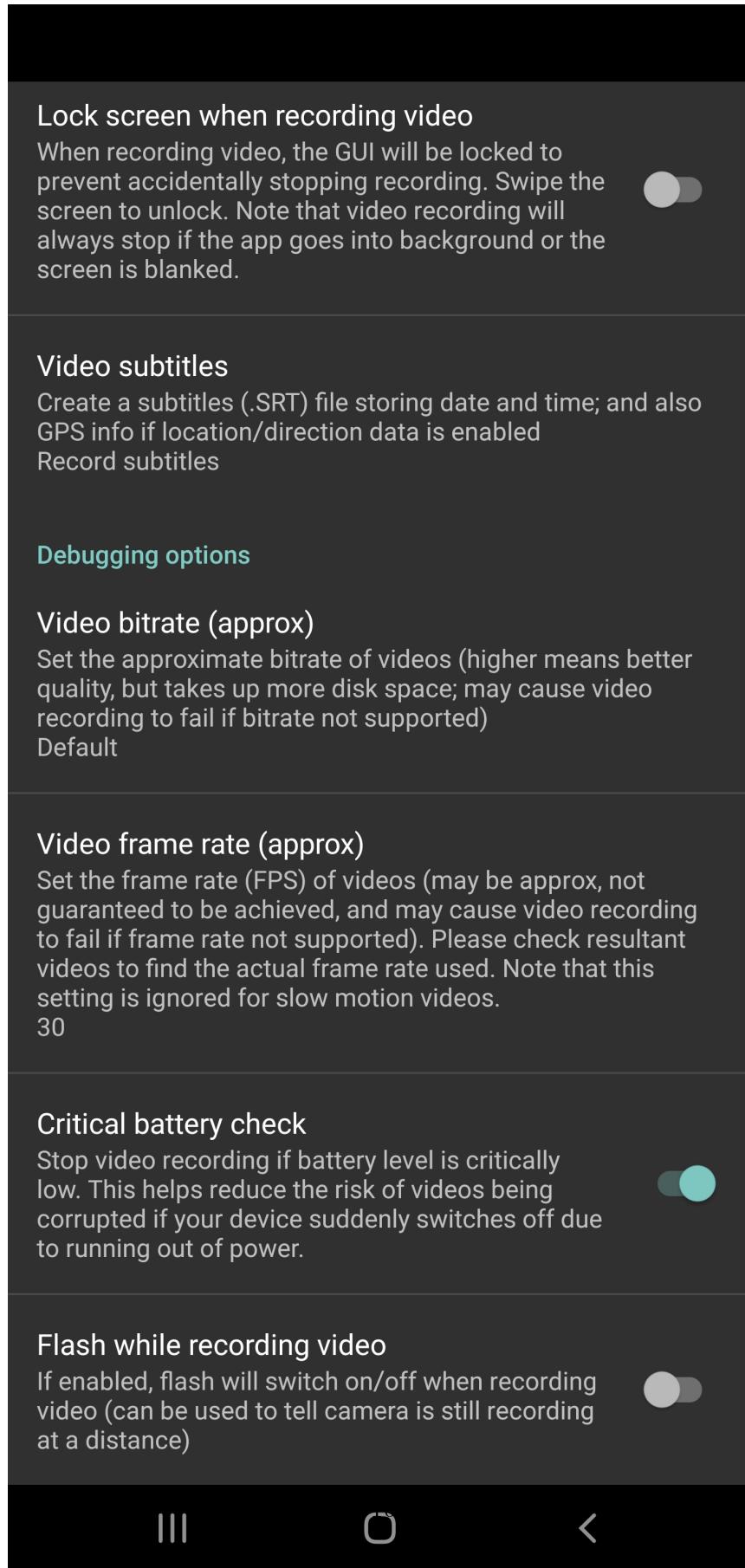


Figure 5: Video settings (3/3).

2.4 Using the GPSLogger App

Although **Open Camera** has an option to georeference video frames, this feature proved unreliable in preliminary tests. As an alternative, it was decided to use a free called **GPSLogger** which logs timestamped GPS coordinates to a file at a frequency of once per second. **GPSLogger** runs continually in background on the phone during a series of video recordings.

2.5 Synchronizing Smart Phone System Time with GPS Time

Georeferencing is done by linking timestamps in the video files recorded by **Open Camera** app with the GPS log generated by the **GPSLogger** app. **Open Camera** uses the phone's internal clock (system time), whereas **GPSLogger** uses time downloaded from GPS satellites (GPS time). System time and GPS time may differ by several minutes, especially if the phone's clock is not regularly synchronized with an internet time server.

There are apps such as **Smart Time Sync** which automatically set the phone's clock to GPS time, but this requires rooting the phone. As an alternative, we use the **GPS Time** app to measure the difference between GPS time and system time. This difference is recorded and used to convert system time to GPS time when running the georeferencing program.

3 Data processing

Data processing is controlled by a python script, process-data.py (Listing 1).

Listing 1: process-data.py

```
"""
This script does great stuff.

Usage:
my_program (--fromdate=<date>) [--todate=<date>] (--datadir=<path>) (--database=<filename>)
my_program -h | --help

Options:
-h --help Show this screen.
--fromdate=<date> First survey date represented as an integer; example: 20201001 (required)
--todate=<date> Last survey date represented as an integer; example: 20201031 (optional)
--datadir=<path> Data directory; example: /home/aubrey/Desktop/Guam01
--database=<filename> Database file name; example: guam01.db

"""

from docopt import docopt
import subprocess
import os
import logging
import glob

def main():

    # Configure logging
    logging.basicConfig(
        level=logging.INFO,
        format="%(asctime)s [%(levelname)s] %(message)s",
        datefmt="%Y-%m-%dT%H:%M:%S%z",
        handlers=[logging.StreamHandler()])
    # Get command line arguments
    args = docopt(__doc__, version='0.1.0')
```

```

args = docopt(__doc__, version='0.1')
fromdate = args.get('--fromdate')
todate = args.get('--todate')
datadir = args.get('--datadir')
database = args.get('--database')
dbpath = f'{datadir}/{database}'

# Get path to python interpreter used by virtual environment
python_bin = '/home/aubrey/Documents/roadside/code/venv/bin/python'

# Create database if it does not already exist
logging.info('Creating database if it does not already exist.')
if os.path.exists(dbpath):
    logging.info(f'{dbpath} already exists.')
else:
    logging.info(f'Creating {dbpath}')
    command= f'spatialite_{dbpath}_<_home/aubrey/Documents/roadside/spatialite/create_tables2.sql'
    subprocess.run(command, shell=True)
    command = f'spatialite_{dbpath}_<_home/aubrey/Documents/roadside/spatialite/create_views.sql'
    subprocess.run(command, shell=True)

# Process directories for specified survey dates
if not todate:
    todate = fromdate
for videodate in range(int(fromdate), int(todate)+1):
    videodatedir = f'{datadir}/{videodate}'
    if os.path.exists(videodatedir):
        logging.info(f'Processing {videodatedir}')

    # Reduce frame rate to 3 FPS for each video. The original video is renamed to *_original.mp4
    logging.info('Reducing frame rates.')
    for video_path in glob.glob(f'{videodatedir}/{videodate}_?????.mp4'):
        new_path = video_path.replace('.mp4', 'original.mp4')
        if os.path.exists(new_path):
            logging.info(f'{new_path} already exists. Skipping FPS reduction.')
        else:
            os.rename(video_path, new_path)
            logging.info(f'{video_path} --> {new_path}')
            command = f'ffmpeg -i {new_path} -filter:v fps=fps=3_{video_path}'
            output = subprocess.run(command, shell=True)
            logging.info(f'output:{output}')
    logging.info('Finished frame rate reduction.')

# Upload video files to S3 (3 FPS files only, run object detection using OnePanel API,
# and finally download CVAT XML files.
if len(glob.glob(f'{videodatedir}/cvat_annotation*.xml'))==0:
    command = f'python detect_objects.py -dd {datadir} -d {videodate}'
    logging.info(command)
    output = subprocess.run(command, shell=True)
    logging.info(output)
else:
    logging.info('Skipping detect_objects.py because {videodatedir} already contains CVAT XML files')

# Break from this loop if this survey date directory does not contain any CVAT XML files.
# Otherwise proceed to populating database tables
if len(glob.glob(f'{videodatedir}/cvat_annotation*.xml'))==0:
    break

# Register videos in videos database table
# Get exif metadata from original videos and save this in a JSON field
command = f'python3_video2db.py --date {videodate} --data-dir {datadir} --db-path {dbpath}'
logging.info(command)
output = subprocess.run(command, shell=True)
logging.info(output)

# Georeference video frames
command = f'python3_georef.py --date {videodate} --data-dir {datadir} --db-path {dbpath}'
logging.info(command)
output = subprocess.run(command, shell=True)
logging.info(output)

# Parse CVAT XML files
# Populates frames, trees and vcuts table.
command = f'python3_cvatxml2db.py --date {videodate} --data-dir {datadir} --db-path {dbpath}'
logging.info(command)
output = subprocess.run(command, shell=True)
logging.info(output)

# Make map

# Create crb-damage grid (mean_damage_index table)
command = f'spatialite_{dbpath}_<_home/aubrey/Documents/roadside/spatialite/create_grid.sql'
logging.info(command)
output = subprocess.run(command, shell=True)
logging.info(output)

```

```

# Create tracks table
command = f'spatialite_{dbpath}<~/home/aubrey/Documents/roadside/spatialite/create_tracks.sql'
logging.info(command)
output = subprocess.run(command, shell=True)
logging.info(output)

# Write $dbpath to a text file so that make_crb_damage_map.py can read it.
command = f'echo_{dbpath}>{dbpath}.txt'
logging.info(command)
output = subprocess.run(command, shell=True)
logging.info(output)

# Deactivate virtual environment
#deactivate

# Create map using QGIS
command = 'qgis --nologo --code make_crb_damage_map.py'
#logging.info(command)
#output = subprocess.run(command, shell=True)
#logging.info(output)

logging.info('***FINISHED ALL***')
logging.info('To see map, deactivate venv, run "qgis --nologo --code make_crb_damage_map.py" and zoom to mean-damage_index_layer.')

if __name__ == '__main__':
    main()

```

3.1 Georeferencing Video Frames

The author has cobbled together a **jupyter notebook** called **georef** which uses the **GPSLogger** log to calculate the GPS coordinates for frames video recordings based on timestamps (see interactive map for displaying coconut rhinoceros beetle roadside video survey routes with popup thumbnail images at points of interest. The prototype map is at <https://aubreymoore.github.io/qgiswebmap/webmap/webmap.ipynb>).

3.2 Mapping

Interactive web maps for displaying coconut rhinoceros beetle roadside video survey routes with popup thumbnail images at points of interest can be put together using QGIS with the QGIS2WEB plugin. A prototype QGIS project is available in a GitHub repo at <https://github.com/aubreymoore/qgiswebmap> and a prototype map is available at <http://github.io/aubreymoore/qgiswebmap/webmap/webmap>.

4 Detecting CRB Damage in Video Frames

5 Creating a CRB Damage Survey Database

Have a look at database diagram (Fig. 6) and code listings 2, 3 and 4.

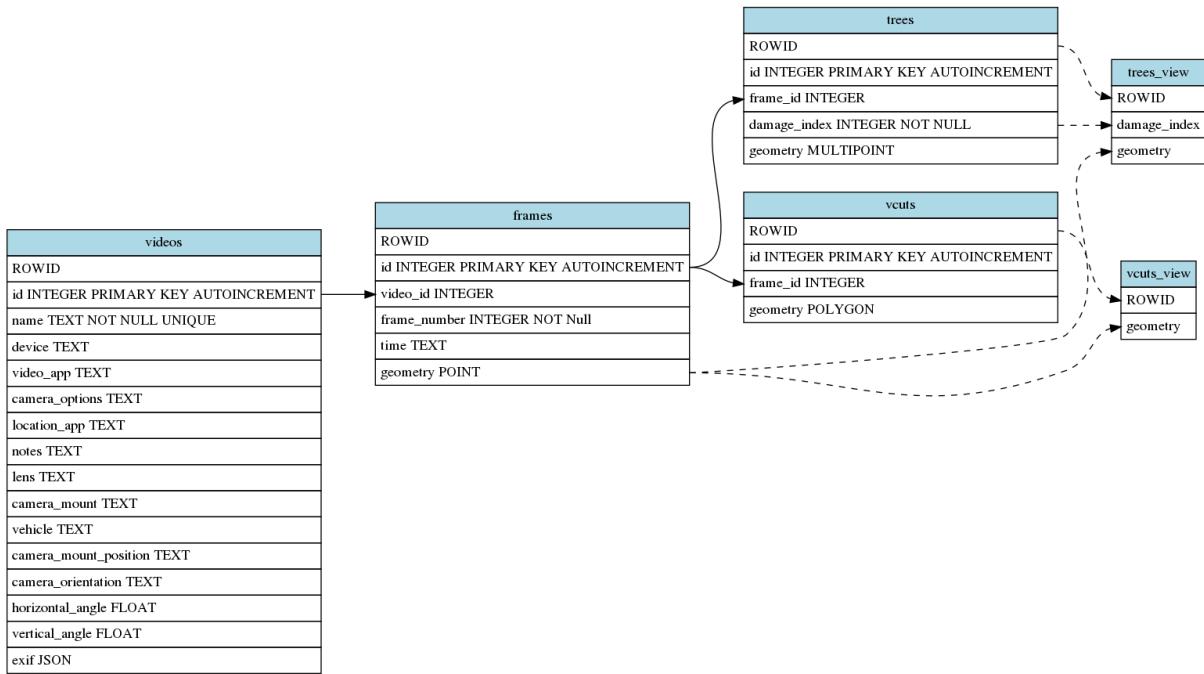


Figure 6: Database diagram.

Listing 2: create_tables.sql

```
-- 
CREATE TABLE videos (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL UNIQUE,
    device TEXT,
    video_app TEXT,
    camera_options TEXT,
    location_app TEXT,
    notes TEXT,
    lens TEXT,
    camera_mount TEXT,
    vehicle TEXT,
    camera_mount_position TEXT,
    camera_orientation TEXT,
    horizontal_angle FLOAT,
    vertical_angle FLOAT,
    exif JSON
);

-- 
CREATE TABLE tracks (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL UNIQUE,
    FOREIGN KEY(name) REFERENCES frames(name)
);

SELECT AddGeometryColumn('tracks', 'geometry', 3857, 'LINESTRING', 'XY');

-- 
CREATE TABLE frames (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    video_id INTEGER,
    frame_number INTEGER NOT Null,
    time TEXT,
    FOREIGN KEY(video_id) REFERENCES videos(id),
    UNIQUE(video_id, frame_number)
);
```

```

SELECT AddGeometryColumn('frames', 'geometry', 3857, 'POINT', 'XY');

-- 
CREATE TABLE trees (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    frame_id INTEGER,
    damage_index INTEGER NOT NULL,
    FOREIGN KEY(frame_id) REFERENCES frames(id)
);

SELECT AddGeometryColumn('trees', 'geometry', -1, 'MULTIPOINT', 'XY');

-- 
CREATE TABLE vcuts (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    frame_id INTEGER,
    FOREIGN KEY(frame_id) REFERENCES frames(id)
);

SELECT AddGeometryColumn('vcuts', 'geometry', -1, 'POLYGON', 'XY');

```

Listing 3: create_views.sql

```

-- Aubrey Moore 2020-10-2021

-- Creates a view for use with QGIS
-- The geometry column contains camera location coordinates.
-- Note: SQL for this spatially enabled view was developed using spatialite_gui Query/View Composer

-- THIS DOES NOT WORK PROPERLY:

CREATE VIEW "trees_view" AS
SELECT "a"."damage_index" AS "damage_index", "b"."ROWID" AS "ROWID", "b"."geometry" AS "geometry"
FROM "trees" AS "a"
JOIN "frames" AS "b" ON ("a"."frame_id" = "b"."id");

INSERT INTO views_geometry_columns
(view_name, view_geometry, view_rowid, f_table_name, f_geometry_column, read_only)
VALUES ('trees_view', 'geometry', 'rowid', 'frames', 'geometry', 1);

-- Creates a view for use with QGIS
-- The geometry column contains camera location coordinates.
-- Note: SQL for this spatially enabled view was developed using spatialite_gui Query/View Composer

CREATE VIEW "vcuts_view" AS
SELECT "b"."ROWID" AS "ROWID", "b"."geometry" AS "geometry"
FROM "vcuts" AS "a"
JOIN "frames" AS "b" ON ("a"."frame_id" = "b"."id");

INSERT INTO views_geometry_columns(
view_name, view_geometry, view_rowid, f_table_name, f_geometry_column, read_only)
VALUES ('vcuts_view', 'geometry', 'rowid', 'frames', 'geometry', 1);

```

Listing 4: create_grid.sql

```

BEGIN;

DROP TABLE IF EXISTS mean_damage_index;

CREATE TABLE grid (id INTEGER PRIMARY KEY AUTOINCREMENT);

SELECT AddGeometryColumn('grid', 'geometry', 3857, 'MULTIPOLYGON', 'XY');

INSERT INTO grid (geometry)
    SELECT SquareGrid(Extent(geometry), 1000) FROM frames;

CREATE TABLE grid1 (id INTEGER PRIMARY KEY AUTOINCREMENT);

SELECT AddGeometryColumn('grid1', 'geometry', 3857, 'POLYGON', 'XY');

INSERT INTO grid1 (geometry)
    SELECT geometry
    FROM ElementaryGeometries
    WHERE f_table_name = 'grid'
        AND origin_rowid=1;

```

```

CREATE TABLE mean_damage_index (id INTEGER PRIMARY KEY AUTOINCREMENT, mean_damage_index DOUBLE);

SELECT AddGeometryColumn('mean_damage_index', 'geometry', 3857, 'POLYGON', 'XY');

INSERT INTO mean_damage_index (mean_damage_index, geometry)
    SELECT AVG(damage_index), grid1.geometry
    FROM trees_view, grid1
    WHERE Contains(grid1.geometry, trees_view.geometry)
    GROUP BY grid1.id;

--Clean up

DROP TABLE grid;
DROP TABLE grid1;

COMMIT;

```

6 Visualizing Survey Results on a Map

QGIS 3.16 was used to visualize survey results on a map.

- The map was made using a python script, **create_crb_damage_map.py**, (Listing ??) which reads data from the spatialite database.
- An interactive web map is then created using the **qgis2web** plugin. See Figures 7 and 8 for **qgis2web** parameters.
- The resulting web map, stored in index.html, modified using **edit_webmap.py** () .
- Finally, index.html was saved in a GitHub repository and published using GitHub pages.

Listing 5: make-crb-damage-map.py

```

# qgis --codepath make_crb_damage_map.py
# Aubrey Moore 2020-10-27
# This script is not yet complete.

# After it has run, zoom to the mean_damage_index layer.

# Use the qgis2web plugin to make an interactive web map
# leaflet; Add layer list: expanded; extnet: fit layers extent; template:full_screen
# Upload /tmp/qgis2web/qgis2web/qgis2web_2020_09_16-06_58_26_614131 to
# https://github.com/aubreymoore/Guam-CRB-damage-map

from qgis.utils import iface
from PyQt5.QtGui import *

def load_guam_osm():
    canvas = iface.mapCanvas()
    url = 'type=xyz&url=https://a.tile.openstreetmap.org/{z}/{x}/{y}.png&crs=EPSG3857'
    rlayer = QgsRasterLayer(url, 'Guam', 'wms')
    QgsProject.instance().addMapLayer(rlayer)
    rect = QgsRectangle(16098000.0, 1486000.0, 16137000.0, 1535000.0)
    canvas.setExtent(rect)
    canvas.update()

def load_layer_from_db(table_name):
    # CANNOT GET DBPATH FROM TEXT FILE???: CHECK LAS CHAR
    # Get dbpath from a text file named dbpath.txt

```

```

with open('dbpath.txt') as f:
    dbpath = f.readline().strip()
    print(f'dbpath: {dbpath}')
#print(f'dbpath: "(dbpath)"')
#dbpath='/home/aubrey/Desktop/Guam01/20201019.db'
print(f'dbpath: {dbpath}')
uri = QgsDataSourceUri()
uri.setDatabase(dbpath)
schema = ''
table = table_name
geom_column = 'geometry'
uri.setDataSource(schema, table, geom_column)
display_name = table_name
vlayer = QgsVectorLayer(uri.uri(), display_name, 'spatialite')
QgsProject.instance().addMapLayer(vlayer)

def style_mean_damage_index():
    mean_damage_index_layer = QgsProject.instance().mapLayersByName(
        'mean_damage_index')[0]
    target_field = 'mean_damage_index'
    legend = [
        {'low': 0.0, 'high': 0.0, 'color': '#008000', 'label': '0.0_no_damage'},
        {'low': 0.0, 'high': 0.5, 'color': '#0ff00', 'label': '0.0_<0.5_very_light_damage'},
        {'low': 0.5, 'high': 1.5, 'color': '#ffff00', 'label': '0.5_<1.5_light_damage'},
        {'low': 1.5, 'high': 2.5, 'color': '#ffa500', 'label': '1.5_<2.5_medium_damage'},
        {'low': 2.5, 'high': 3.5, 'color': '#ff6400', 'label': '2.5_<3.5_high_damage'},
        {'low': 3.5, 'high': 4.0, 'color': '#ff0000', 'label': '3.5_<4.0_moribund/dead'},
    ]
    myRangeList = []
    for i in legend:
        symbol = QgsSymbol.defaultSymbol(mean_damage_index_layer.geometryType())
        symbol.setColor(QColor(i['color']))
        symbol.setOpacity(0.75)
        myRangeList.append(QgsRendererRange(i['low'], i['high'], symbol, i['label'], True))
    myRenderer = QgsGraduatedSymbolRenderer(target_field, myRangeList)
    myRenderer.setMode(QgsGraduatedSymbolRenderer.Custom)
    mean_damage_index_layer.setRenderer(myRenderer)

def style_tracks_layer():
    tracks_layer = QgsProject.instance().mapLayersByName('tracks')[0]
    tracks_layer.renderer().symbol().setWidth(1)
    tracks_layer.renderer().symbol().setColor(QColor(255,0,0))

def style_vcuts_view_layer():
    vcuts_view_layer = QgsProject.instance().mapLayersByName('vcuts_view')[0]
    vcuts_view_layer.renderer().symbol().setColor(QColor(255,0,255))

def style_tree_location_layer():
    c0 = QgsRendererCategory(0,
        QgsMarkerSymbol.createSimple({'color': '#00ff00'}),
        "0_No_damage",
        True)
    c1 = QgsRendererCategory(1,
        QgsMarkerSymbol.createSimple({'color': '#ffff00'}),
        "1",
        True)
    c2 = QgsRendererCategory(2,
        QgsMarkerSymbol.createSimple({'color': '#ffa500'}),
        "2",
        True)
    c3 = QgsRendererCategory(3,
        QgsMarkerSymbol.createSimple({'color': '#ff6400'}),
        "3",
        True)
    c4 = QgsRendererCategory(4,
        QgsMarkerSymbol.createSimple({'color': '#ff0000'}),
        "4",
        True)
    renderer = QgsCategorizedSymbolRenderer("damage_index", [c0,c1,c2,c3,c4])
    vlayer = QgsProject.instance().mapLayersByName('tree_location')[0]
    vlayer.setRenderer(renderer)

def show_feature_counts():
    root = QgsProject.instance().layerTreeRoot()
    for child in root.children():
        if isinstance(child, QgsLayerTreeLayer):
            child.setCustomProperty("showFeatureCount", True)

def add_abstract():

```

```

QgsProject.instance().metadata().setAbstract('placeholder')

# MAIN

load_guam_osm()
load_layer_from_db('mean_damage_index')
load_layer_from_db('tracks')
load_layer_from_db('vcuts_view')
load_layer_from_db('tree_location')
style_mean_damage_index()
style_tracks_layer()
style_vcups_view_layer()
style_tree_location_layer()
show_feature_counts()
add_abstract()

```

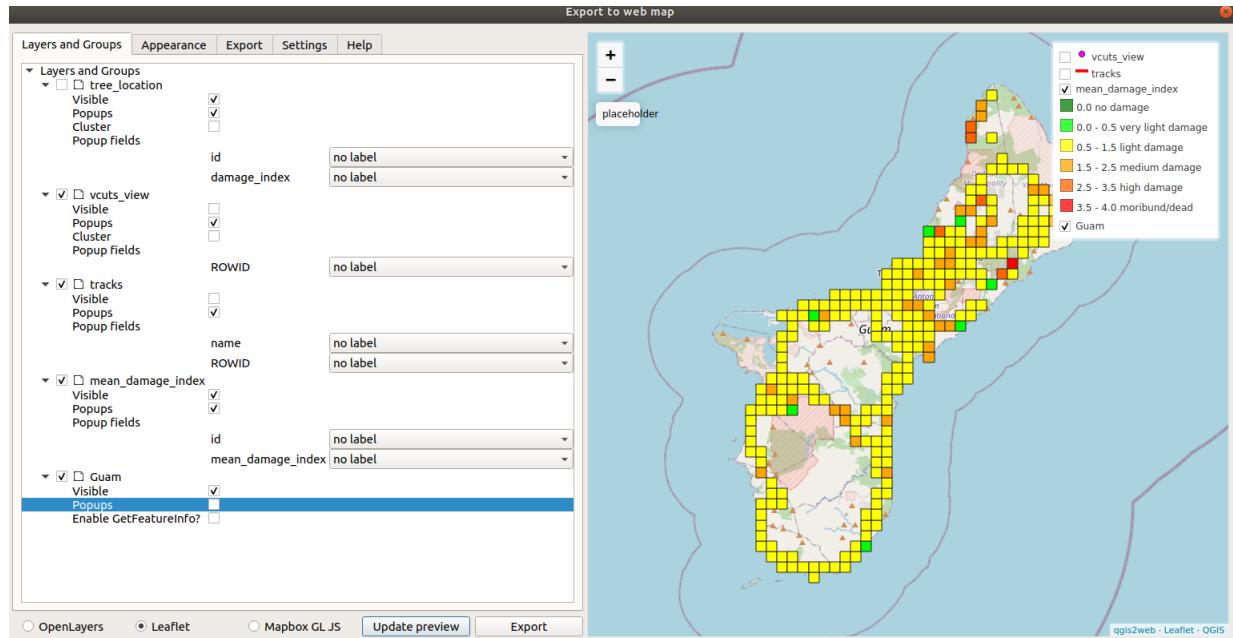


Figure 7: **qgis2web** parameters Screenshot 1.

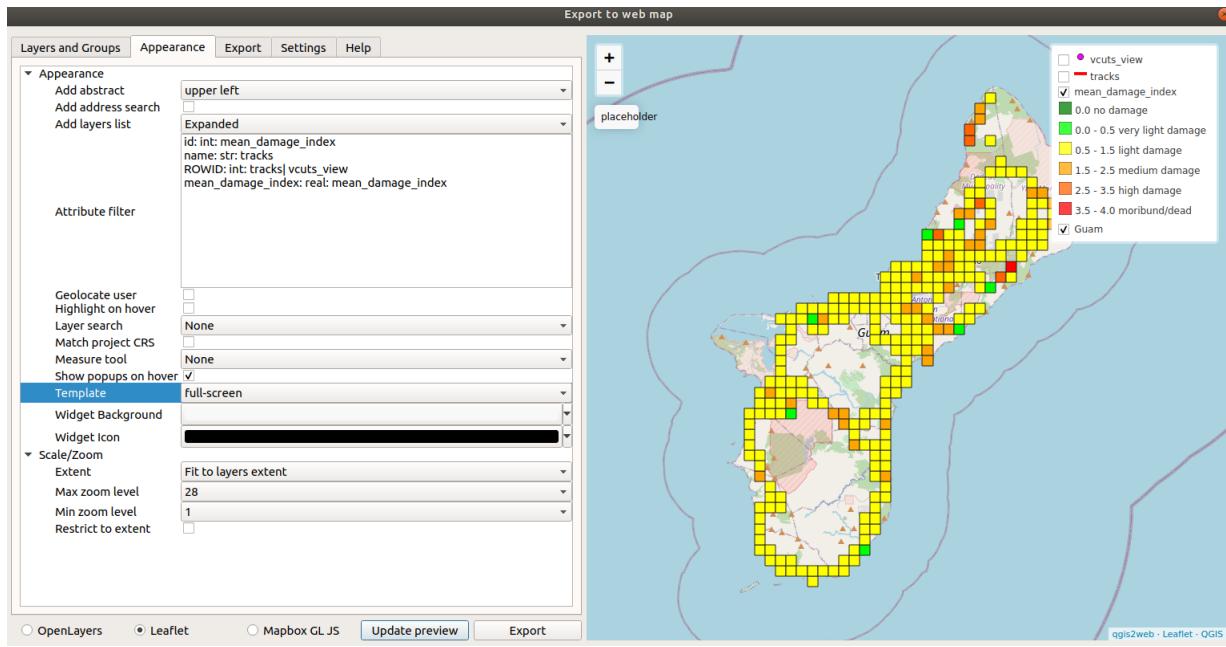


Figure 8: `qgis2web` parameters. Screenshot 2.

7 First Roadside Video Survey of CRB Damage on Guam

7.1 Methods

	Development	Guam01 survey
Vehicle	Mazda Miata	Toyota Forerunner
Cell phone	Samsung Galaxy S10	Samsung Galaxy S8
Lens	Ultra-wide angle	Standard angle
Frame rate	30 FPS	15 FPS

7.2 Results

Data and results from the Guam01 survey are available in a public GitHub repository [Moore 2020a](#). An interactive web map is available on GitHub pages (Fig. 9).

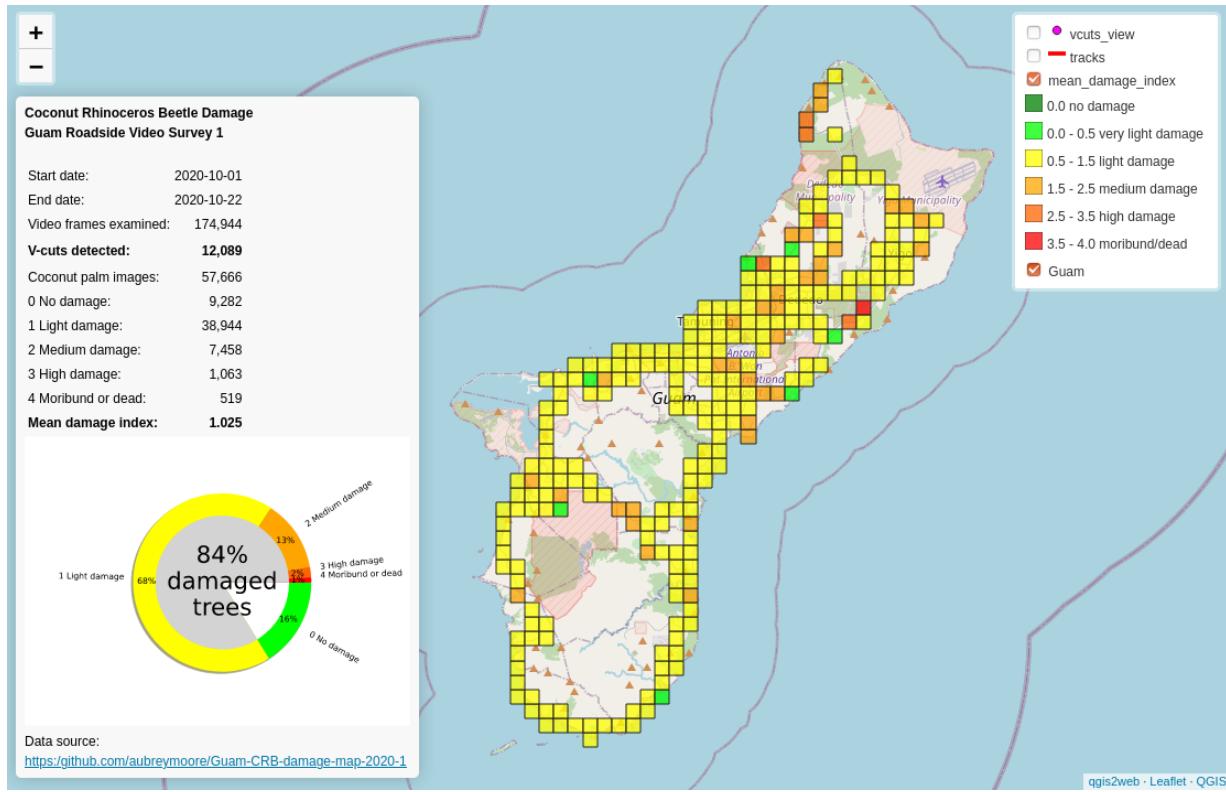


Figure 9: Screenshot of an interactive web map of results from the Guam01 roadside video survey of CRB damage on Guam (Moore 2020b) URL: <https://aubreymoore.github.io/Guam-CRB-damage-map-2020-10>.

8 Extra

Aubrey Moore 2020
OnePanel Inc. 2020a
OnePanel Inc. 2020b

9 Test Data

The following creates a 30 second video clip starting at 10 minutes and 30 seconds after the start of the video. Start time = $10:46:54 + 00:10:30 = 10:57:24$.

```
mv 20210304_104654.mp4 20210304_104654_original.mp4
ffmpeg \
  -ss 00:10:30 \
  -i 20210304_104654_original.mp4 \
  -ss 00:00:30 \
  -t 00:00:30 \
  -c copy \
  20210304_104654.mp4
```

Unfortunately, timestamps in the resulting file are not set, so we have to edit using:

```
exiftool "-CreateDate=2021:03:04 10:57:24" 20210304_101635.mp4
```

10 References

Aubrey Moore (Oct. 26, 2020). *Set Up for Automated Roadside Video Surveys for Detecting and Monitoring Coconut Rhinoceros Beetle Damage on Rota*. URL: <https://api.zotero.org/groups/511387/items/3X83KK58/file/view>.

Jackson, Trevor A. (May 20, 2019). “Rhinoceros Beetle Damage. Workshop with Kokonas Indastri Koporesen (KIK). Madang, PNG”. In: URL: <https://api.zotero.org/groups/511387/items/QMD5TZQU/file/view>.

Moore, Aubrey (2020a). *GitHub Repository: Guam-CRB-Damage-Map-2020-10*. URL: <https://github.com/aubreymoore/Guam-CRB-damage-map-2020-10>.

Moore, Aubrey (2020b). *Web Map: Guam-CRB-Damage-Map-2020-10*. URL: <https://aubreymoore.github.io/Guam-CRB-damage-map-2020-10>.

OnePanel Inc. (July 24, 2020a). *AI Pipeline Operations Manual: University of Guam - CRB Damage Pipeline v.1.0*. URL: <https://api.zotero.org/groups/511387/items/DXCX92JC/file/view>.

OnePanel Inc. (June 4, 2020b). *Scope of Work: Object Detector(s) for Quantification of Coconut Rhinoceros Beetle Damage in Roadside Video Surveys V1.1*. URL: <https://api.zotero.org/groups/511387/items/8ZSUF6BX/file/view>.

Vaqalo, Maclean, Visoni Timote, Senimili Baiculacula, Gideon Suda, and Frank Kwainarara (June 2017). *The Coconut Rhinoceros Beetle in the Solomon Islands: A Rapid Damage Assessment of Coconut Palms on Guadalcanal*. URL: <https://api.zotero.org/groups/511387/items/LJDIQZYA/file/view>.