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

Aubrey Moore 2020-10-18

1

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

# Contents

1	Bac	ckground	3	
2	Recording Roadside Videos			
	2.1	Mounting a Smart Phone on a Vehicle	3	
		2.1.1 Setting the Direction of View Angle	3	
	2.2	Parameter Choices	3	
		2.2.1 Camera and Lens	3	
		2.2.2 Resolution	3	
		2.2.3 Frames per Second	4	
	2.3	Using the Open Camera App	4	
	2.4	Using the GPSLogger App	4	
	2.5	Georeferencing Video Frames	4	
	2.6	Mapping	4	
3	Det	ecting CRB Damage in Video Frames	5	
4	Cre	ating a CRB Damage Survey Database	5	
5	Vici	ualizing Survey Results on a Map	5	
J	VISC	danzing Survey Results on a Map	J	
6	Ref	erences	5	
L	isti	ngs		
			_	
	1	create_tables.sql	6	
	2	create_views.sql	7	
	3	create_grid.sql	7	
	4	create_crb_damage_map.py	8	
ı	ist	of Figures		
	156	or i iguies		
	1	Smart phone mount	9	
	2	Setting camera angles using Clinometer app	10	
	3	Video settings $(1/3)$	11	
	4	Video settings $(2/3)$	12	
	5	Video settings $(3/3)$	13	
	6	Database diagram.	14	

# 1 Background

To see how coconut rhinoceros beetle damage surveys are currently done, please read Jackson 2019 and Vagalo 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).

#### 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.

#### 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.

## 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 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 (seeinteractive map for displaying coconut rhinoceros beetle roadside video survey routes with popup thumbnail images at points of interest. The prototype map is at <a href="https://aubreymoore.github.io/qgiswebmap/webmap/webmap">https://github.com/aubreymoore/roadside/blob/master/jupyter%20notebooks/georef.ipynb</a>).

# 2.6 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.

# 3 Detecting CRB Damage in Video Frames

# 4 Creating a CRB Damage Survey Database

Have a look at database diagram (Fig. 6) and code listings 1, 2 and 3.

# 5 Visualizing Survey Results on a Map

# 6 References

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.

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.

## Listing 1: 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,
       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 vouts (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
   frame_id INTEGER,
   FOREIGN KEY(frame_id) REFERENCES frames(id)
SELECT AddGeometryColumn('vcuts', 'geometry', -1, 'POLYGON', 'XY');
```

#### Listing 2: create views.sql

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

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 spatiallite_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_column, read_only)

VALUES ('vcuts_view', 'geometry', 'rowid', 'frames', 'geometry', 1);
```

#### Listing 3: create grid.sql

```
BEGIN:
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;
```

#### Listing 4: create\_crb\_damage\_map.py

```
# qgis --codepath make_crb_damage_map.py
# This script is not yet complete.
# After it gas 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
{\tt\#\ https://github.com/aubreymoore/Guam-CRB-damage-map}
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 get dbpath():
    with open ('dbpath.txt') as f:
      dbpath = f.read()
    return dbpath
def load_layer_from_db(dbpath, table_name):
    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 = [
        gend = [
{'low': 0.0, 'high': 0.0, 'color': '#008000', 'label': 'No_damage'},
{'low': 0.0, 'high': 0.5, 'color': '#00ff00', 'label': '0.0__0.5'},
{'low': 0.5, 'high': 1.5, 'color': '#ffff00', 'label': '0.5__1.5'},
{'low': 1.5, 'high': 2.5, 'color': '#ff6500', 'label': '1.5__2.5'},
{'low': 2.5, 'high': 3.5, 'color': '#ff6400', 'label': '2.5__3.5'},
{'low': 3.5, 'high': 4.0, 'color': '#ff6000', 'label': '3.5__4.0'},
    myRangeList = []
    for i in legend:
        symbol = QgsSymbol.defaultSymbol(mean_damage_index_layer.geometryType())
        symbol.setColor(QColor(i['color']))
        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 laver = OgsProject.instance().mapLaversByName('tracks')[0]
    tracks_layer.renderer().symbol().setWidth(1)
    {\tt 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))
# MAIN
load_guam_osm()
dbpath = get_dbpath()
load_layer_from_db(dbpath, 'tracks')
load_layer_from_db(dbpath, 'mean_damage_index')
load_layer_from_db(dbpath, 'vcuts_view')
style_mean_damage_index()
style_tracks_layer()
style vcuts view layer()
```

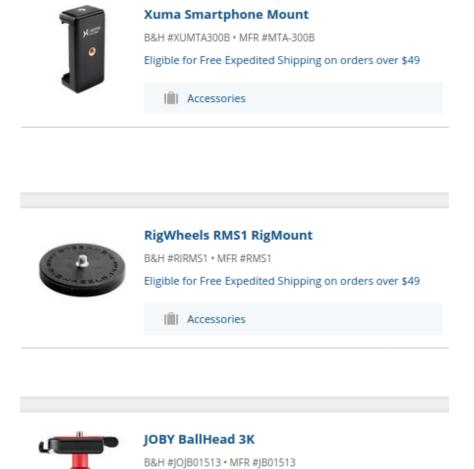


Figure 1: Smart phone mount.

Eligible for Free Expedited Shipping on orders over \$49

Free Shipping for this Item

Accessories



Figure 2: Setting camera angles using Clinometer app.

## Video resolution

4K 3840x2160 (16:9, 8.29MP)

#### Enable digital video stabilization

Video stabilization reduces the shaking due to the motion of the camera in both the preview and in recorded videos. This may be unnecessary if your device supports optical image stabilization (OIS).



#### Video format

Video and audio file format and codecs Default

#### Video picture profiles

Set standard or flat picture profile for video mode Default

#### Video gamma value

Gamma value to use for video if video picture profile is set to Gamma

#### Maximum duration of video

The video recording will stop after the specified duration Unlimited

#### Restart video after max duration

If the video has stopped after hitting the maximum duration (if set), this option will make the video restart, up to the specified number of times

Don't restart

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

Figure 4: Video settings (2/3).

#### Lock screen when recording video

When recording video, the GUI will be locked to prevent accidentally stopping recording. Swipe the screen to unlock. Note that video recording will always stop if the app goes into background or the screen is blanked.



#### Video subtitles

Create a subtitles (.SRT) file storing date and time; and also GPS info if location/direction data is enabled Record subtitles

#### **Debugging options**

#### Video bitrate (approx)

Set the approximate bitrate of videos (higher means better quality, but takes up more disk space; may cause video recording to fail if bitrate not supported) Default

### Video frame rate (approx)

Set the frame rate (FPS) of videos (may be approx, not guaranteed to be achieved, and may cause video recording to fail if frame rate not supported). Please check resultant videos to find the actual frame rate used. Note that this setting is ignored for slow motion videos.

## Critical battery check

Stop video recording if battery level is critically low. This helps reduce the risk of videos being corrupted if your device suddenly switches off due to running out of power.



## Flash while recording video

If enabled, flash will switch on/off when recording video (can be used to tell camera is still recording at a distance)



III



<

Figure 5: Video settings (3/3).

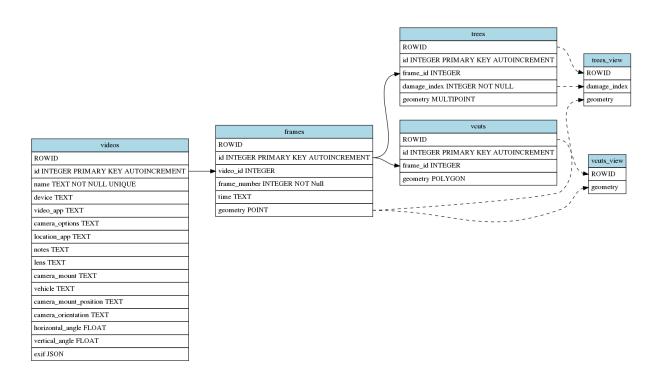


Figure 6: Database diagram.