# Step 1.2 Creating photo-interpreted plots

## 1.2.1 Correcting Observed map with recent FIELD PLOTS

Diane ESPEL

2025-06-18

## Contents



# 1 Objectives

This script aims to correct and complete the attribute values of polygons in a vector map using geolocated field quadrats.

To achieve this:

- The polygon layer `"observed_map"` will be intersected with the field survey layer `"field_samples"`
- The attribute values of the intersected polygons will be replaced by the dominant attribute value (if multiple surveys exist) from `"field_samples"`
- The polygons to be corrected are those that intersect with a ground truth layer.

# 2 Script explanation

## 2.1 Clean environment and graphics

```
rm(list = ls())  # Clear all objects from the R environment to start fresh
graphics.off()  # Close all graphics devices (if any plots are open)
```

## 2.2 Load required packages

```
library(sp)  # Classes and methods for handling spatial data (legacy package)
library(sf)  # Simple Features for spatial data - modern and efficient spatial data package
library(terra)  # Raster and vector spatial data manipulation and analysis
library(dplyr)  # Data manipulation functions like filter, join, select, mutate, etc.
```

## 2.3 Create functions

We need to define:

- A "`get_class_dom()`" function that returns the most frequent habitat class : This function identifies the dominant habitat class within a vector of class labels. It counts the frequency of each class and determines which one(s) occur most frequently. If there is a single most frequent class, it returns that class. If there is a tie between three or more classes, it returns the first one. If exactly two classes are tied, it returns the second one. The output is the dominant habitat class according to these rules.
- A "`transform_chain()`"function that normalizes the nomenclature of habitats: here, adding dots to the toponymy (e.g., A11 –> A.1.1): This function formats a string by inserting dots between each character. It splits the input string into individual characters, joins them using dots, and returns the resulting formatted string. This is useful for visually separating characters in habitat codes.

```
# Function to find the most frequent habitat class(es) and return the dominant
# one
get_class_dom <- function(vector_classes) {

    # Count frequency of each habitat class in the input vector
    classes_freqs <- table(vector_classes)

    # Find the highest frequency count
    max_freq <- max(classes_freqs)

    # Identify all classes with the highest frequency
    max_categories <- names(classes_freqs)[classes_freqs == max_freq]

    # Return dominant class based on number of max frequency classes: - If only
    # one class, return it - If three or more classes tie, return the first one
    # - If exactly two classes tie, return the second one
    if (length(max_categories) == 1) {
        classdom <- max_categories
    } else if (length(max_categories) >= 3) {
        classdom <- max_categories[1]
    } else {
        classdom <- max_categories[-1]
```

```
    }

    # Return the dominant habitat class
    return(classdom)
}

# Function to insert dots between each character in a string (for formatting
# habitat codes)
transform_chain <- function(chain) {

    # Split string into individual characters
    divided_chain <- strsplit(chain, "")[[1]]

    # Join characters by inserting dots between them
    transformed_chain <- paste(divided_chain, collapse = ".")

    # Return the transformed string with dots
    return(transformed_chain)
}
```

## 2.4 Define Global Variables

Note: A global variable is a variable defined outside of any function. This means the variable is accessible from any part of the code, including inside functions. A global variable retains its value throughout the execution of the R script unless it is explicitly modified in the code.

It is important to define at a minimum:

- the `"District"`: the archipelago of interest (e.g. "CRO" for Crozet archipelago)
- the `"Island"`: the island within the archipelago of interest (e.g. "POS" for Possession island)
- the `"Satellite1"`: the name of the satellite used for photo-interpretation
- the `"Year1"`: the year of imagery acquisition (i.e. the photo-interpretation year)

```
District = "CRO"  # 3-letter code for the archipelago
Island = "POS"  # 3-letter code for the island
Satellite1 = "Pleiades"  # Name of satellite used for photo-interpretation
Year1 = "2022"  # Year of imagery acquisition
```

## 2.5 Set working directory

You must define a general root directory (`"localHOME"`) that serves as the base path for your input and output data. This directory should point to the local environment where: - input sample plots file is located under `"data/vector/Plots/PrimaryTypo"` - input polygon map of observed habitats is located under `"data/vector/Observed_map/PrimaryTypo"` - output new polygon map will be saved under `"data/vector/Observed_map/PrimaryTypo"`

```
# Base local path (customize to your local environment) localHOME =
# paste0('your_local_path/')
localHOME = paste0("/home/genouest/cnrs_umr6553/despel/CARTOVEGE/")

# Path to open input plot vector data and observed map vector data
open_plots_path = paste0(localHOME, "data/vector/Plots/PrimaryTypo")
open_ObsMap_path = paste0(localHOME, "data/vector/Observed_map/PrimaryTypo")
```

```r
# Path to save corrected observed maps
save_ObsMap_path = paste0(localHOME, "data/vector/Observed_map/PrimaryTypo")
```

## 2.6 Load files

```r
# Load input spatial data
print("We want to correct the photo-interpreted map with FIELD SAMPLES")

# Compose file paths for polygon map and field samples shapefiles
FILE1 = paste0(open_ObsMap_path, "/", "Observed_map_", District, "_", Island, "_",
    Satellite1, "_", Year1, "_EPSG32739.shp")
FILE2 = paste0(open_plots_path, "/", "Quadrats_", District, "_", Island,
 ↪  "_ALL_FIELD_SAMPLES_Polygons_corrected_EPSG32739.shp")

# Read shapefiles into sf objects
observed_map <- st_read(FILE1)
field_samples <- st_read(FILE2)

# Check for duplicate IDs in field samples and polygons map
print("There should be no duplicate IDs")
colnames(field_samples)[colnames(field_samples) == "ID"] <- "id"  # Rename ID column for
 ↪  consistency
id_doublons_field <- which(duplicated(field_samples$id) | duplicated(field_samples$id,
    fromLast = TRUE))
print(id_doublons_field)  # Print any duplicate IDs found in field samples
id_doublons_map <- which(duplicated(observed_map$id) | duplicated(observed_map$id,
    fromLast = TRUE))
print(id_doublons_map)  # Print any duplicate IDs found in polygons map
```

## 2.7 Intersect FIELD_SAMPLES and OBSERVED_MAP

The polygon layer is intersected by the quadrat layer to find polygons to correct/update. To achieve this:

- This code first checks whether the coordinate reference systems (CRS) of two spatial datasets match (`st_crs()`).
- If not, it transforms (`st_transform()`) the observed_map to the CRS of `field_samples`.
- It then computes the spatial intersection between the polygons in `observed_map` and the `field_samples`, returning the overlapping areas.

```r
print("Identify FIELD samples falling within polygons of the map")

# Check and match coordinate reference systems (CRS)
st_crs(OBSERVED_MAP) == st_crs(field_samples)

# Transform observed_map to the CRS of field_samples if different
observed_map <- st_transform(observed_map, st_crs(field_samples))

# Compute spatial intersection between polygons and field samples
intersect_polys <- st_intersection(observed_map, field_samples)

print(paste0("Number of FIELD samples located within map polygons: ", nrow(intersect_polys)))
```

## 2.8 Correct intersected polygons data frame using dominant FIELD samples values

This script assigns a dominant habitat class to each polygon in a spatial map based on the field samples it contains:

- It ensures the spatial object is correctly formatted.
- For each polygon, it identifies all overlapping field samples.
- If multiple field sample plots are present within a polygon, the dominant class, defined as the most frequent class among these plots, will be identified using the `"get_class_dom()"`
- This dominant class is assigned to the polygon and reformatted to match habitat typology codes.
- Habitat codes are then split hierarchically into levels.
- Finally, the dataset is trimmed to retain only relevant columns.

```
intersect_polys_corr = intersect_polys

# Confirm class is sf object
class(intersect_polys_corr)
intersect_polys_corr <- st_as_sf(intersect_polys_corr)  # Ensure it's an sf object

# Find dominant habitat class of FIELD samples inside each polygon of
# observed_map
print("Finding the dominant habitat class of FIELD_SAMPLES within each polygon")

# Initialize Dominant_class column as NA
intersect_polys_corr$Dominant_class <- NA

# Identify column indices for id and habitat classification
idmap = which(colnames(intersect_polys_corr) == "id")  # ID column index
ih4sample = which(colnames(intersect_polys_corr) == "Hab_L4.1")  # Habitat level 4 (sample) column
 ↪  index

# Extract unique polygon IDs from intersected data
ID_list <- unique(intersect_polys_corr[[idmap]])

# Loop over each polygon ID
for (i in ID_list) {

    # Select samples belonging to polygon i
    print(paste0("Retrieving all samples present in polygon ", i, " of the map"))
    polygon_of_interest <- intersect_polys_corr[intersect_polys_corr[[idmap]] ==
        i, ]
    polygon_of_interest <- st_as_sf(polygon_of_interest)

    # Determine dominant habitat class among samples in polygon i
    print("Finding dominant class among selected samples")
    dominant_class <- get_class_dom(polygon_of_interest[[ih4sample]])

    # Assign the dominant class to all samples of polygon i
    intersect_polys_corr[intersect_polys_corr[[idmap]] == i, "Dominant_class"] <- dominant_class[1]
}

# Format Dominant_class habitat codes to match observed_map typology (insert
# dots)
```

```r
intersect_polys_corr$Dominant_class <- sapply(intersect_polys_corr$Dominant_class,
    transform_chain)

# Update habitat classification columns based on dominant class
ih1map = which(colnames(intersect_polys_corr) == "Hab_L1")
ih2map = which(colnames(intersect_polys_corr) == "Hab_L2")
ih3map = which(colnames(intersect_polys_corr) == "Hab_L3")
ih4map = which(colnames(intersect_polys_corr) == "Hab_L4")

intersect_polys_corr$New_hab = intersect_polys_corr$Dominant_class

# Assign progressively detailed habitat codes to columns Hab_L1 through Hab_L4
intersect_polys_corr[, ih1map] = substr(intersect_polys_corr$New_hab, 1, 1)  # Level 1 code (first
↪  character)
intersect_polys_corr[, ih2map] = substr(intersect_polys_corr$New_hab, 1, 3)  # Level 2 code (first
↪  3 characters)
intersect_polys_corr[, ih3map] = substr(intersect_polys_corr$New_hab, 1, 5)  # Level 3 code (first
↪  5 characters)
intersect_polys_corr[, ih4map] = intersect_polys_corr$New_hab  # Level 4 code (full string)

# Keep only columns up to 'id' column to trim data frame
idmap = which(colnames(intersect_polys_corr) == "id")
intersect_polys_corr = intersect_polys_corr[, 1:idmap]

print("Corrected dataframe of intersected polygons is ready")
```

## 2.9 Correct the original observed_map with updated classification from intersected polygons

This script updates the habitat classification of each polygon in the observed map based on previously computed dominant classes:

- It creates a copy of the original map to preserve the raw data.
- For each polygon ID, it retrieves the corresponding corrected classification.
- If data exists, it updates the habitat levels (L1 to L4) in the copied map.
- Finally, it saves the corrected map as a shapefile using standardized naming.

```r
print("Correcting the observed_map using the corrected intersected polygons dataframe")

# Create a copy of observed_map for correction
observed_map_corr <- observed_map

# Loop over each polygon ID to update habitat classes
for (i in ID_list) {

    print(paste0("Retrieving all samples present in polygon ", i, " of the map"))

    # Extract corrected polygon data for polygon i
    corrected_polygon <- intersect_polys_corr[intersect_polys_corr[[idmap]] == i,
        ]

    if (nrow(corrected_polygon) > 0) {
        # Update habitat classification columns in observed_map_corr with
```

```
        # corrected values
        observed_map_corr[observed_map_corr[[idmap]] == i, c("Hab_L1", "Hab_L2",
            "Hab_L3", "Hab_L4")] <- corrected_polygon[1, c("Hab_L1", "Hab_L2", "Hab_L3",
            "Hab_L4")]
    }
}

# Save the corrected Observed map to disk as a shapefile
st_write(observed_map_corr, paste0(save_ObsMap_path, "/", "Corrected_observed_map_",
    District, "_", Island, "_", Satellite1, "_", Year1, "_EPSG32739.shp"), driver = "ESRI
 ↪  Shapefile",
    append = FALSE)
```

This script enables the refinement/updating of photo-interpreted maps based on ground true field data.