Supplementary Materials

Supplementary Material S1

From 14 orthomosaics we manually recognized 100 black-necked swans individuals. Then we extracted each pixel from each and determined the value for each color band. We considered the minimum critical value of each band's the upper value of the first quartile of the pixel distribution. We considered as potential swans each value equal or superior to this critical value. The color setting for each swan was 220 for the red band, 221 for the green band, and 221 for the blue band (Figure 1). Each orthomosaics was filtered according to the previous configuration, generating a binary raster (1, pixels that meet the condition; 0, pixels that do not meet the condition).

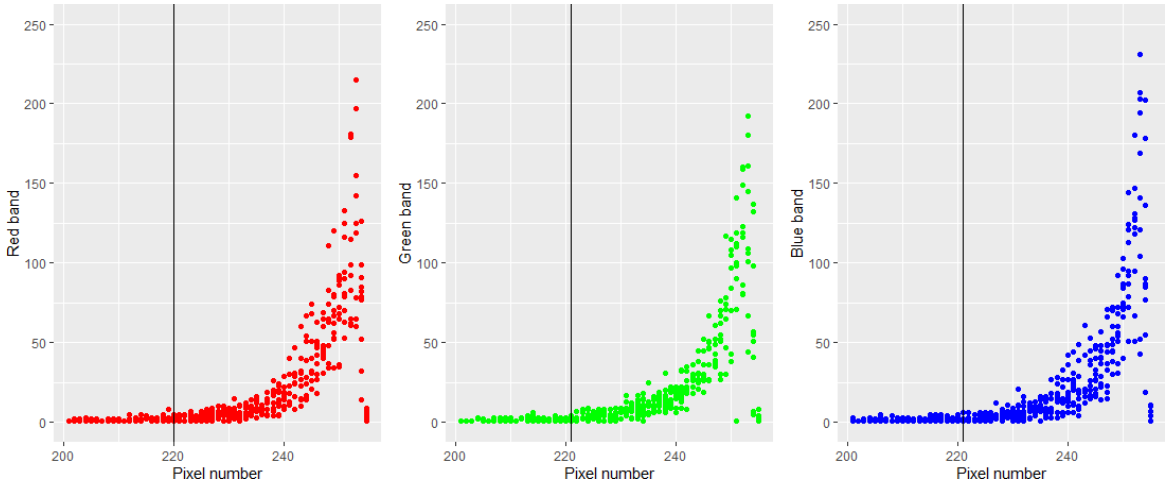

Figure S1. Pixel value distribution of 100 swans. The x-axis represents the band value for each pixel (red, green and blue), the y-axis represents the total number of pixels for each band value. The black line parallel to the y-axis represents the upper limit of the first quartile of the distribution, values to the right of the line are considered as potential swans.

Supplementary Material S2

R-code for Automatic Recognition of Black-necked Swans (*Cygnus melancoryphus*) from Orthophotographs

The main objective of this investigation is to standardize a methodology for automatic bird counting from orthophotographs, to be used in the long-term monitoring of the Black-necked Swan, *Cygnus melancoryphus.* using R software

```
#packages installation
install.packages("raster","rgdal","geosphere","spatstat","maptools","gdalUtils","rgeos","spatialEco","R
.utils")

#load libraries
library("raster","rgdal","geosphere","spatstat","maptools","gdalUtils","rgeos","spatialEco","R.utils")
```

In a first instance, an object containing the orthophoto to be analyzed, must be created. The *stack()* function of the raster package is used, since the orthophoto is a TIFF extension image composed of three red, green and blue color bands. This step will generate a *Formal class RasterStack* object.

```
orthophotography<-stack("orthophotography.tif")

#class       : RasterStack
#dimensions  : 25473, 23528, 599328744, 4  (nrow, ncol, ncell, nlayers)
#resolution  : 3.830408e-07, 3.830409e-07  (x, y)
#extent      : -73.27503, -73.26602, -39.82073, -39.81098  (xmin, xmax, ymin, ymax)
#coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
#names       : ortofotografia.1, ortofotografia.2, ortofotografia.3, ortofotografia.4
#min values  :              0,              0,              0,              0
#max values  :            255,            255,            255,            255
```

When the orthophoto is loaded, the first step is to make a thick filter that separates the background of the orthophoto from the pixels of interest. In this case, we make a filter according to the color of the black-necked swans. Section 2.4.1. describes how to determine the value to be used.

```
#[[X]] corresponds to the band number to be filtered
# >= 220 select all pixels greater than and equal to the value 220

color.filter <- orthophotography[[1]]>=220 & orthophotography[[2]]>=221 & orthophotography[[3]]
>=221

#class       : RasterLayer
#dimensions  : 25473, 23528, 599328744  (nrow, ncol, ncell)
#resolution  : 3.830408e-07, 3.830409e-07  (x, y)
#extent      : -73.27503, -73.26602, -39.82073, -39.81098  (xmin, #xmax, ymin, ymax)
#coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
#names       : layer
#values      : 0, 1  (min, max)
```

This generates a *RasterLayer* object with two values, 0 for pixels that did not match the filtering condition (background of the orthophoto), and 1 for pixels that match the condition (black-necked swans). After this step, the *color.filter* object is vectorized, so you have to load the *polygonizer()* function to be able to carry out the vectorization.

```
#Vectoring function [1]

polygonizer <- function(x, outshape=NULL, pypath=NULL, readpoly=TRUE,
                        quietish=TRUE) {
  if (isTRUE(readpoly)) require(rgdal)
  if (is.null(pypath)) {
    cmd <- Sys.which('OSGeo4W.bat')
    pypath <- 'gdal_polygonize'
    if(cmd=='') {
      cmd <- 'python'
      pypath <- Sys.which('gdal_polygonize.py')
      if (!file.exists(pypath))
        stop("Could not find gdal_polygonize.py or OSGeo4W on your system.")
    }
  }
  if (!is.null(outshape)) {
    outshape <- sub('\\.shp$', '', outshape)
    f.exists <- file.exists(paste(outshape, c('shp', 'shx', 'dbf'), sep='.'))
    if (any(f.exists))
      stop(sprintf('File already exists: %s',
                   toString(paste(outshape, c('shp', 'shx', 'dbf'),
                                  sep='.')[f.exists])), call.=FALSE)
  } else outshape <- tempfile()
  if (is(x, 'Raster')) {
  require(raster)
    writeRaster(x, {f <- tempfile(fileext='.tif')})
    rastpath <- normalizePath(f)
  } else if (is.character(x)) {
    rastpath <- normalizePath(x)
  } else stop('x must be a file path (character string), or a Raster object.')

  system2(cmd, args=(
    sprintf('"%s" "%s" %s -f "ESRI Shapefile" "%s.shp"',
            pypath, rastpath, ifelse(quietish, '-q ', ''), outshape)))

  if (isTRUE(readpoly)) {
    shp <- readOGR(dirname(outshape), layer = basename(outshape),
                   verbose=!quietish)
    return(shp)
  }
  return(NULL)
}
```

The *filtercolor* object is vectorized with the *polygonizer()* function generating an object of class *SpatialPolygonsDataFrame,* where each space object of value 1 (DN=1) represents a potential black-necked swan individual.

```
# Vectorize the filtered orthomosaic
polygon <-polygonizer(color.filter)

#class       : SpatialPolygonsDataFrame
#features    : 252
#extent      : -73.27501, -73.26605, -39.82068, -39.81101  (xmin, xmax, ymin, ymax)
#coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
#variables   : 1
#names    :   DN
#min  values  :  0
#max values  :  1
```

Then, the polygon layer is assigned a projection, for this example the WGS 84 18S projection was used. To do this, it is necessary to create an object that contains the CRS that you want to assign to the polygon() object and then use the function *spTransform()*

```
#Assign CRS to the vectored layer
#Establish the new object with CRS information
crs.geo <- CRS("+proj=utm +zone=18 +south +datum=WGS84 +units=m +no_defs +ellps=WGS84
+towgs84=0,0,0")

#Change the CRS for the data
polygon<- spTransform(polygon, crs.geo)


#class       : SpatialPolygonsDataFrame
#features    : 252
#extent      : 647652, 648402.8, 5590712, 5591791  (xmin, xmax, ymin, ymax)
#coord. ref. : +proj=utm +zone=18 +south +datum=WGS84 +units=m +no_defs +ellps=WGS84
+towgs84=0,0,0
#variables   : 1
#names       : DN
#min values  :  0
#max values  :  1
```

The space objects with value 0 (DN=0), correspond to polygons that did not fulfill the filtering condition, to the orthophoto contour polygon and to areas without information. Therefore, they were eliminated by creating a subset that did not contain them.

```
polygon<- subset(polygon, DN > 0)
```

Then, different shape parameters were determined for each space object and filtered according to the values corresponding to black-necked swans (see section 2.4.2)

```
#Number of vertices of each space object
polygon$vertices<- as.numeric(sapply(polygon@polygons, function(y)
nrow(y@Polygons[[1]]@coords)))


#filtered according to the number of vertices
polygon <- subset polygon, vertices >= 11 & vertices <= 72)

#Area of each space object
polygon $area <- round(gArea(polygon, byid=TRUE),8)
```

```
#filtered according to the area of the space object
polygon <- subset (polygon, area >= 0.0 1115624 & area <= 0.5501689)

#Perimeter of each space object
polygon$perimeter <- round(polyPerimeter(polygon),8)

#filtered according to the perimeter of each object
polygon <- subset (polygon, perimeter >= 0.6220029 & perimeter <= 4.80825)

#Shape index of each space object
polygon$shape<- (4*pi* polygon $area)/(polygon $perimeter* polygon $perimeter)
polygon <- subset (polygon, shape >= 0.2166254 & shape <= 0.698119)

#Filtered according to the ratio of the area and perimeter of each space object
polygon$ap <- round((polygon $area/ polygon $perimeter),8)
polygon <- subset (polygon, ap >= 0.01793599 & ap <= 0.1476506)
```

Then for each space object the smallest rectangle that can contain it was determined, and then the length and width of that rectangle was calculated. To do this, the *spatial_boxes()* function was used

```
#spatial boxes function
spatial_bboxes <- function(polygons) {
  individual_bb <- function(polygon, projection) {
    polygon <- sp::SpatialPolygons(list(polygon), proj4string = projection)
    spatial_bbox <- as(raster::extent(polygon), "SpatialPolygons")
    spatial_bbox <- spatial_bbox@polygons[[1]]
    spatial_bbox@ID <- polygon@polygons[[1]]@ID
    return(spatial_bbox)
  }
  polys <- lapply(polygons@polygons, individual_bb, polygons@proj4string)
  spatial_polys <- sp::SpatialPolygons(polys, proj4string = polygons@proj4string)
  spatial_polys_df <- sp::SpatialPolygonsDataFrame(spatial_polys, polygons@data)
  return(spatial_polys_df)
}

# calculate bounding box for each space object
bbox<-spatial_bboxes(polygon)

#class       : SpatialPolygonsDataFrame
#features    : 153
#extent      : 647721.5, 648366, 5590775, 5591743  (xmin, xmax, ymin, ymax)
#coord. ref. : +proj=utm +zone=18 +south +datum=WGS84 +units=m +no_defs +ellps=WGS84
+towgs84=0,0,0
#variables   : 7
#names       : DN, vertices, area, perimeter, shape, ap, ID
#min values  :  1,       11, 0.01115624, 0.62200294, 0.233340245003434, 0.01793599,   1
#max values  :  1,       46, 0.29424227, 3.01485744, 0.622116831501786, 0.09759741, 153
```

The bbox layer contains the coordinates of each vertex of the rectangle for each space object, the object v1 and v2 determine the distance of the object on each axis (x and y). The maximum value will be the length of the rectangle and the minimum value the width.

```
v1<-(sapply(bbox@polygons,function(p)max(p@Polygons[[1]]@coords[1:5,1]))-
sapply(bbox@polygons,function(p)min(p@Polygons[[1]]@coords[1:5,1])))
```

```
v2<-(sapply(bbox@polygons,function(p)max(p@Polygons[[1]]@coords[1:5,2]))-
sapply(bbox@polygons,function(p)min(p@Polygons[[1]]@coords[1:5,2])))

#Create a data frame from the data
measures<-as.data.frame(cbind(v1,v2))

# select the maximum distance as the length of the object
length<-apply(measures,1,max)

# select the minimum distance as the width of the object
width<-apply(measures,1,min)

bbox$l length <- length
bbox$width<-width
bbox$bboxarea<- round(gArea(bbox, byid=TRUE),8)
```

Then, the data frame of the polygon and bbox layers are joined by the ID attribute. and the missing filters are made.

```
polygon$ID<-seq(1, n, by=1)
polygon1<-merge(polygon,bbox,by="ID")

#Area of intersection between the area of the box and the area of each swan
polygon1$inter<- (100* polygon1$area)/ polygon1$ bboxarea

#Relationship of the length and width of the box
polygon1$l.w<- polygon1$length/ polygon1$width

#Filtered according to the length, width, length-width ratio and intersection area of each space
object

polygon1<-subset (polygon1, length >= 0.2144787 & length <= 1.187679)
polygon1<-subset (polygon1, width >= 0.136718 & width <= 0.8679846)
polygon1<-subset (polygon1, l.w >= 1.000104 & l.w <= 3.711947)
polygon1<-subset (polygon1, polygon1$inter>= 30.72449 & polygon1$inter <= 88.37884)
polygon1

#class       : SpatialPolygonsDataFrame
#features    : 142
#extent      : 647721.5, 648366, 5590775, 5591743  (xmin, xmax, ymin, ymax)
#coord. ref. : +proj=utm +zone=18 +south +datum=WGS84 +units=m +no_defs +ellps=WGS84
+towgs84=0,0,0
#variables   : 12
#names       : DN, vertices,       area,  perimeter, shape, ap,  ID, length, width, bboxarea, inter, l.w
#min values  :  1,        13,  0.0153386, 0.75315321, 0.269615722978266, 0.02036584,   1,
0.215117463842034, 0.138554213917814, 0.03508641, 39.5086432818222, 1.00891564681247
#max values  :  1,        46, 0.29424227, 3.01485744, 0.622116831501786, 0.09759741, 153,
1.08527691394556,  0.50385909806937, 0.45935088, 81.5555158475663, 3.38456070833851

#Save the polygon1 layer
writeOGR(obj=polygon1 ,dsn=".", layer="recognizer_swans",
        driver="ESRI Shapefile")
```

The remaining space objects in the polygon layer1 correspond to the black-necked swan individuals identified by the recognizer.