

gopro max stills 2 kmz converter

Nunzio Knerr

Robert Godfree

2024-11-15

Table of contents

TL;DR:	1
Description of the Workflow	1
Check & Install Required Packages	2
Set User Options	3
Rename Files	4
Function to calculate distances between image geo-locations.	6
Call Function Above	8
Add Overlays to the Images	10
Generate kml File	13
Convert .kml & Images into a .kmz File	15

TL;DR:

This workflow creates a .kmz file from geocoded 360° images taken with a GoPro Max. It involves package setup, file renaming, selecting images by distance (20m default), adding overlays (optional), creating a .kml file, then finally converting everything to a .kmz file for use in the [pannotator](#) package for annotating.

Description of the Workflow

This workflow has been developed to allow easy creation of .kmz files from 360 degree panospheric images. These can be taken with a gopro Max camera or most consumer drones like those made by DJI. The overlay may need to be adjusted depending on the size of the images generated etc.

Any geocoded equirectangular images (jpegs) can be used, regardless of how they were created, but this workflow is specifically tailored to the gopro Max.

Before using this script we recommend making a backup of the original camera files just in case as this script edits the files directly.

The workflow code below is as follows:

1. Check and install required packages
2. Set user options:
 - > folder containing 360 degree images (jpgs)
 - > distance between images (metres)
 - > add overlays (True/False)
 - > overlay file to use (png with transparency)
3. Rename the files (only renames files if names are 12 characters long)
4. Get subset of images a specified distance apart (defaults to 20m)
5. Add overlays to the images (optional)
6. Create a google earth .kml file
7. Convert the kml file and associated images into a single .kmz file

The resulting .kmz file can then be used in the [pannotator](#) package for annotating.

Check & Install Required Packages

In order for this workflow to function as expected there are a few dependent packages to install and configure. If you are on linux you may have some issues with the 'magick' package, check the documentation [here](#).

```
1 dependentPackages <-  
2   c("rstudioapi",  
3     "stringr",  
4     "tools",  
5     "exiftoolr",  
6     "geosphere",  
7     "stringr",  
8     "gpx",  
9     "magick",  
10    "fs",  
11    "magrittr",
```

```

12     "plotKML",
13     "zip",
14     "usefun"
15 )
16
17 for (i in dependentPackages) {
18   print(paste0("Checking for: ", i))
19
20   # First check if you have the package installed
21   check_for_package <- system.file(package = i)
22   print(check_for_package)
23
24   # If not run the following code to install it.
25   if (check_for_package == "") {
26     print(paste0(i, " package not found .....installing now"))
27     install.packages(i)
28   } else {
29     print(paste0(i, " package is already installed"))
30   }
31 }

```

Set User Options

Here we set the user options which will be used in the following code chunks

```

1 # Choose the directory path containing 360 images.
2 directory <- rstudioapi::selectDirectory(caption = "Select Directory
  ↳ Containing 360 images", label = "Select", path =
  ↳ rstudioapi::getActiveProject())
3 # OR
4 # add path manually: use forward slashed only.
5 # directory <- "C:/FolderContaining360Images/"
6
7 # Select the minimum distance in metres between each extracted image.
8 # For example, set metresBetweenEachImageWanted <- 100 to extract
  ↳ images a minimum of 100 metre apart.
9
10 metresBetweenEachImageWanted <- 100
11
12 # Set to TRUE to add overlays to each image file; or FALSE to use
  ↳ images without any overlays.

```

```

13 addOverlays <- TRUE
14
15 # Conditionally set the overlay image file if addOverlays is TRUE
16 if (addOverlays) {
17   # Choose the file for your desired overlay image (PNG image with
18   ↪ transparency).
19   overlayImageFile <- rstudioapi::selectFile(caption = "Select overlay
20   ↪ file (png)", label = "Select PNG file", path =
21   ↪ rstudioapi::getActiveProject(), filter = "png (*.png)",)
22   # OR
23   # specify file path manually.
24   # overlayImageFile <-
25   ↪ "overlay_files/5m_overlay_wedges_straight6.png"
26 }

```

Rename Files

By default most consumer cameras like the gopro max & DJI drones don't allow the user to specify the file names they apply to images that they create.

A typical file name follows the format GS__XXXX.JPG - where XXXX is a counter number of the images taken by the camera.

To address this issue and make it easier to manage the files for processing, this code prepends the date_time stamp to the beginning of the files in a given directory. It's useful for organising files when doing field work, especially when using multiple cameras at the same time.

The output format is: YYYYMMDD_HHMMSS_FileName.ext

Note: Gopro now have a custom firmware allowing you to set file names in the field; see this [GoPro Labs link](#).

This code checks the file name length initially assuming that files names directly downloaded from the camera are 12 characters long. If the files used have longer file names they will not be renamed. This ensures they are only renamed once.

```

1 library(exiftoolr)
2 library(stringr)
3 library(tools)
4
5 # Check if 'directory' is set and valid, throw an error if not

```

```

6  if (!exists("directory") || !dir.exists(paste0(directory))) {
7    stop("'directory' does not exist. Please run the code chunk under
    ↪ 'Set User Options' above to set the directory containing the 360
    ↪ images.")
8  }
9
10 # filter only .jpg or .JPG files
11 file_extension <- "\\.[Jj][Pp][Gg]$"
12
13 my_files <-
14   list.files(
15     directory,
16     pattern = paste0("*. ", file_extension),
17     all.files = FALSE,
18     full.names = TRUE
19   )
20
21 #read the exif information in the file to get the creation date
22 files_df <- exiftoolr::exif_read(my_files, args = c("-G1", "-a",
    ↪ "-s"))
23
24 #Loop through the files and check to change file names
25 #this checks if the files have already been changed by looking at the
    ↪ length of the file name.
26 for (i in 1:nrow(files_df)) {
27   print("Checking if camera file name has not been changed")
28   if (nchar(files_df[i, "System:FileName"]) == 12) {
29     print("File appears to be 12 characters long")
30     print(paste0("SourceFile: ", files_df[i, "SourceFile"]))
31     origFullFileName <- paste0(files_df[i, "SourceFile"])
32     createDate <- paste0(files_df[i, "ExifIFD:DateTimeOriginal"])
33     print(paste0("CreateDate: ", createDate))
34     formattedCreateDate <- stringr::str_replace_all(createDate, ":",
    ↪ "")
35     formattedCreateDate <-
    ↪ stringr::str_replace_all(formattedCreateDate, " ", "_")
36     print(paste0("formattedCreateDate: ", formattedCreateDate))
37     file_ext <- tolower(tools::file_ext(files_df[i,
    ↪ "System:FileName"]))
38     newFileName <- paste0(files_df[i, "System:Directory"], "/",
    ↪ formattedCreateDate, "_", tools::file_path_sans_ext(basename(files_df[i,
    ↪ "System:FileName"])), ".", file_ext)

```

```

39   print(paste0("newFileName: ", newFileName))
40   file.rename(from = origFullFileName, to = newFileName)
41   print("File name changed")
42 } else {
43   print(
44     "It appears that the file has already been renamed as it's
      ↪ greater than 12 characters long"
45   )
46   print(paste0("SourceFile: ", files_df[i, "SourceFile"]))
47 }
48
49 }

```

Function to calculate distances between image geo-locations.

This code looks through all the files in a given folder and copies images a user-specified distance apart into a new folder for use later on. It starts with the first file and looks for a file at least XX metres from that. Once it finds one it adds it to the list then uses it as the location to look for another file at least XX metres from it and so on until it gets to the end of the file list. This method is most suitable for linear transect sampling but should work with any images that are spaced out enough.

```

1  library(geosphere)
2
3  options(digits = 20)
4  options(digits.secs = 20)
5  options(scipen = 9999)
6
7  #function which takes 2 arguments
8  #1:gpx_locations - a dataframe containing 4 columns("SourceFile",
      ↪ "System:Directory", "Composite:GPSLongitude",
      ↪ "Composite:GPSLatitude")
9  #2:distance in metres between each image to extract. (default=20m)
10 findImagesEveryXmetres <-
11   function(my_gpx_locs, metresToNextImage = 20) {
12     gpx_locs <- my_gpx_locs
13
14     keeps <- c("Composite:GPSLongitude", "Composite:GPSLatitude")
15     points <- gpx_locs[keeps]
16

```

```

17 #View(points)
18 #View(gpx_locs)
19
20 #calculate the distance between any two points
21 distance_m <- geosphere::distm(points , fun =
↪ geosphere::distHaversine)
22 rownames(distance_m) <- basename(gpx_locs[, "SourceFile"])
23 colnames(distance_m) <- basename(gpx_locs[, "SourceFile"])
24
25 #View(distance_m)
26
27 #find images a certain distance apart.
28 selected_files <- vector()
29
30 metres_between_images <- metresToNextImage
31
32 print(paste0(
33   "Searching for images apart by: ",
34   metres_between_images,
35   " metres"
36 ))
37
38 for (i in 1:nrow(distance_m)) {
39   if (i == 1) {
40     #if it is the first frame add it as the current frame
41     selected_files <-
42       append(selected_files, rownames(distance_m)[i])
43     current_frame <- rownames(distance_m)[i]
44     print(paste0("Frame 1: ", current_frame))
45     print(paste0(
46       "looking for frame >",
47       metres_between_images ,
48       " Metres from frame 1"
49     ))
50   }#if the current frame is greater than the specified metres
51   if ((distance_m[i, current_frame] > metres_between_images)) {
52     current_frame <- rownames(distance_m)[i]
53     print(paste0("current_frame: ", current_frame))
54     selected_files <- append(selected_files, current_frame)
55   }
56

```

```

57   }
58   print(paste0("Files found:", selected_files))
59
60   new_folder <-
61     paste0(gpx_locs[1, "System:Directory"], "_Frames_",
62           ↪ metres_between_images, "m_apart")
63
64   dir.create(new_folder)
65
66   source_folder <-  dirname(gpx_locs[1, "SourceFile"])
67
68   print(gpx_locs[1, "System:Directory"])
69
70   for (q in selected_files) {
71     file_to_copy <- paste0(source_folder, "/", q)
72     destination <- paste0(new_folder, "/", q)
73     file.copy(
74       file_to_copy,
75       destination,
76       overwrite = TRUE,
77       recursive = FALSE,
78       copy.mode = TRUE,
79       copy.date = TRUE
80     )
81   }
82 }

```

Call Function Above

Now call the function above to calculate the distance between all the images and copy them to a new folder.

```

1 library(exiftoolr)
2
3 # Check if 'directory' is set and valid, throw an error if not
4 if (!exists("directory") || !dir.exists(paste0(directory))) {
5   stop("'directory' does not exist. Please run the code chunk under
6     ↪ 'Set User Options' above to set the directory containing the 360
7     ↪ images.")

```



```

6 }
7
8 file_extension <- "\\.[Jj][Pp][Gg]$"
9
10 my_files <-
11   list.files(
12     directory,
13     pattern = paste0(file_extension),
14     all.files = FALSE,
15     full.names = TRUE
16   )
17
18 image_files_df <-
19   exiftoolr::exif_read(my_files, args = c("-G1", "-a", "-s"))
20
21 #View(image_files_df)
22
23 gpx_locs <-
24   as.data.frame(image_files_df[, c(
25     "SourceFile",
26     "System:Directory",
27     "Composite:GPSLatitude",
28     "Composite:GPSLongitude"
29   )])
30
31 #View(gpx_locs)
32
33 if (!exists("metresBetweenEachImageWanted")) {
34   print("'metresBetweenEachImageWanted' does not exist. Using Default
35     ↪ value. Please run the code chunk under 'Set User Options' above
36     ↪ if you want to change the metresBetweenEachImageWanted")
37   findImagesEveryXmetres(my_gpx_locs = gpx_locs)
38 } else {
39   findImagesEveryXmetres(my_gpx_locs = gpx_locs, metresToNextImage =
40     ↪ metresBetweenEachImageWanted)
41 }

```

Add Overlays to the Images

This code goes through the images in the folder created above and adds the overlay file to them. This overlay must be specific to the camera used to create the 360 degree images as the focal length of the lens etc. will define how the overlay should look.

In this example we used a gopro Max at 3.2m above the ground. The easiest way to determine how an overlay should look is to take some images with the camera at the specified height with the desired overlay marked on the ground so you have an easy template to base your overlay on.

Here we wanted a circular marker with a 5 metre radius and we were lucky to find a round concrete water tank buried in the ground with the required radius. We marked the distance in metres from the centre of the plot directly under the camera using a pole with black marking tape at 1 metre intervals. Below is the image loaded into [inkscape](#) so we could draw the required marker lines for the overlay.

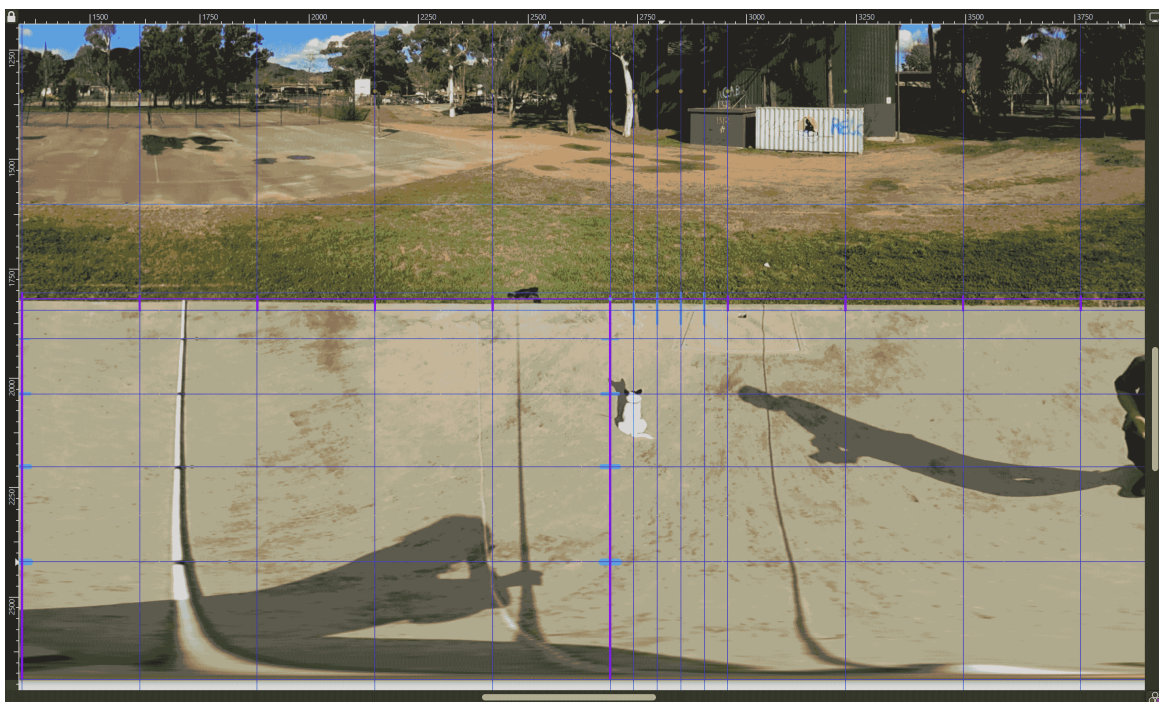


Figure 1: overlay image with camera background

Note: There is a slight discrepancy with the line on the right side of the image. This is due to the camera not being exactly vertical when capturing the image.

The overlay was created using inkscape and then exported as a portable network graphics (.png) file with transparency. See the example below:

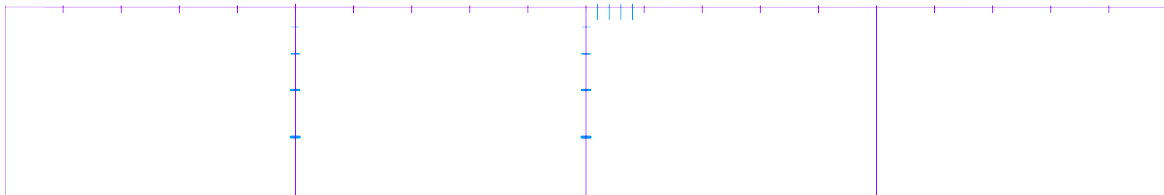


Figure 2: overlay image with transparency

The code below uses [imagemagick](#) to load the underlying base file and then overlays the .png and saves out the flattened file for use in the kml/kmz files in the following steps. If you are on linux and have issues with the 'magick' package you may need to up the memory settings in the /etc/ImageMagick-6/policy.xml file.

```
<policy domain="resource" name="memory" value="4GiB"/>
<policy domain="resource" name="map" value="8GiB"/>
<policy domain="resource" name="disk" value="16GiB"/>
<policy domain="resource" name="area" value="10GiB"/>
```

You can find more info on how to do this [here](#).

```
1 library(magick)
2 library(tools)
3
4 # Check if 'directory' is set and valid, throw an error if not
5 if (!exists("directory") || !dir.exists(paste0(directory))) {
6   stop("'directory' does not exist. Please run the code chunk under
7     ↳ 'Set User Options' above to set the directory containing the 360
8     ↳ images."
9   )
10 }
```

```

10 # if 'metresBetweenEachImageWanted' doesn't exist then add the default
    ↪ metres between images
11 if (!exists("metresBetweenEachImageWanted")) {
12   print("'metresBetweenEachImageWanted' not selected...using default:
    ↪ Please run the code chunk under 'Set User Options' above to set
    ↪ the metresBetweenEachImageWanted."
13   )
14   metresBetweenEachImageWanted <- 20
15 }
16
17 # if 'addOverlays' doesn't exist then throw an error asking to set
    ↪ 'addOverlays'
18 if (!exists("addOverlays")) {
19   stop("'addOverlays' does not exist. Please run the code chunk under
    ↪ 'Set User Options' above to set the addOverlays.")
20 }
21
22 if (addOverlays == TRUE) {
23   overlay_file <- overlayImageFile
24
25   new_directory <- paste0(directory,
26                           "_Frames_",
27                           metresBetweenEachImageWanted,
28                           "m_apart")
29
30   if (!dir.exists(paste0(new_directory))) {
31     print(paste0(new_directory, " does not exist!"))
32     stop("Did you run the code chunk above to find images a certain
    ↪ distance apart?"
33   )
34 }
35
36 # first create a new directory to add the overlay images to
    dir.create(paste0(new_directory, "/with_overlay/"))
37
38 file_extension <- "\\.[Jj][Pp][Gg]$"
39
40 files_lst <-
41   list.files(
42     new_directory,
43     pattern = paste0(file_extension),
44     all.files = FALSE,

```

```

45     full.names = TRUE,
46     recursive = FALSE,
47     include.dirs = FALSE
48   )
49
50
51   for (t in 1:length(files_lst)) {
52     background_image <- magick::image_read(files_lst[t])
53     overlay <-
54       magick::image_read(overlay_file)
55     image_dir <- dirname(files_lst[t])
56     overlay_image_dir <- paste0(image_dir, "/with_overlay/")
57     new_filename <-
58       paste0(overlay_image_dir,
59             basename(tools::file_path_sans_ext(files_lst[t])),
60             "_with_overlay.jpg")
61     print(paste0("Adding overlay to create: ", new_filename))
62     img <- c(background_image, overlay) %>%
63       magick::image_flatten(.) %>%
64       magick::image_write(., new_filename, format = "jpg")
65   }
66
67 } else {
68   print("'addOverlays' not TRUE: No overlay files generated")
69 }

```

Generate kml File

This code generates a [google earth](#) kml file linking to the image files in the folder generated above. It uses [ExifTool](#) with a template “kml_hide_rollover.fmt” to create the kml file.

```

1 # Check if 'directory' is set and valid, throw an error if not
2 if (!exists("directory") || !dir.exists(paste0(directory))) {
3   stop("'directory' does not exist. Please run the code chunk under
4     ↳ 'Set User Options' above to set the directory containing the 360
5     ↳ images."
6   )
7 }
8
9 # if 'metresBetweenEachImageWanted' doesn't exist then add the default
10 ↳ metres between images

```

```

8  if (!exists("metresBetweenEachImageWanted")) {
9    print("'metresBetweenEachImageWanted' not selected...using default:
    ↪ Please run the code chunk under 'Set User Options' above to set
    ↪ the metresBetweenEachImageWanted."
10   )
11   metresBetweenEachImageWanted <- 20
12 }
13
14 if (addOverlays == TRUE) {
15
16 new_directory <-
17   paste0(directory,
18     "_Frames_",
19     metresBetweenEachImageWanted,
20     "m_apart/with_overlay")
21
22 output_kml <-
23   paste0(directory,
24     "_Frames_",
25     metresBetweenEachImageWanted,
26     "m_apart_with_overlay.kml")
27 } else if(addOverlays == FALSE) {
28   new_directory <-
29     paste0(directory,
30       "_Frames_",
31       metresBetweenEachImageWanted,
32       "m_apart")
33
34 output_kml <-
35   paste0(directory,
36     "_Frames_",
37     metresBetweenEachImageWanted,
38     "m_apart.kml")
39 }
40
41 exif_args <- c("-p", "kml_hide_rollover.fmt", "-r")
42 exif_call(
43   args = exif_args,
44   path = new_directory,
45   stdout = output_kml,
46   quiet = FALSE

```

```

47 )
48
49 # now fix the links to the images to make them relative.
50 mystring <- readr::read_file(output_kml)
51 path_only <- paste0(dirname(output_kml))
52 mystring2 <- gsub(path_only, ".", mystring, fixed = T)
53
54 # Write the file out
55 sink(paste0(output_kml))
56   writeLines(mystring2)
57 sink()
58
59 print(paste0("generated kml file: ", output_kml))

```

Convert .kml & Images into a .kmz File

This code reads the .kml file created above and converts it to a .kmz file. This involves zipping up the images and the .kml file into one file. It also edits the relative links etc. The convenience of the kmz file is that it combines the kml and associated images into one file.

NOTE: This code can generate kmz files >2GB. These files won't open correctly in google earth but are not corrupt and will work fine in pannotator. This is a limitation of google earth being 32 bit. You can read about it [here](#).

```

1 library(zip)
2
3 # Check if 'directory' is set and valid, throw an error if not
4 if (!exists("directory") || !dir.exists(paste0(directory))) {
5   stop("'directory' does not exist. Please run the code chunk under
6     ↳ 'Set User Options' above to set the directory containing the 360
7     ↳ images."
8   )
9 }
10
11 # if 'metresBetweenEachImageWanted' doesn't exist then add the default
12   ↳ metres between images
13 if (!exists("metresBetweenEachImageWanted")) {
14   print("'metresBetweenEachImageWanted' not selected...using default:
15     ↳ Please run the code chunk under 'Set User Options' above to set
16     ↳ the metresBetweenEachImageWanted."

```

```

12   )
13   metresBetweenEachImageWanted <- 20
14 }
15
16 if (addOverlays == TRUE) {
17
18 new_directory <-
19   paste0(directory,
20     "_Frames_",
21     metresBetweenEachImageWanted,
22     "m_apart/with_overlay")
23
24 output_kml <-
25   normalizePath(paste0(directory,
26     "_Frames_",
27     metresBetweenEachImageWanted,
28     "m_apart_with_overlay.kml"), winslash = "/", mustWork =
29     ↪ FALSE)
30 } else if (addOverlays == FALSE) {
31   new_directory <-
32     paste0(directory,
33       "_Frames_",
34       metresBetweenEachImageWanted,
35       "m_apart")
36
37 output_kml <-
38   normalizePath(paste0(directory,
39     "_Frames_",
40     metresBetweenEachImageWanted,
41     "m_apart.kml"), winslash = "/", mustWork = FALSE)
42 }
43
44 print("Generating kmz file for:")
45 print(output_kml)
46
47 kml_file_name <- basename(output_kml)
48 kml_image_directory <- new_directory
49
50 dir_to_copy <- normalizePath(kml_image_directory, winslash = "/",
51   ↪ mustWork = FALSE)
52 temp_folder <- paste0(usefun::get_parent_dir(directory), "/temp")

```



```

51 new_dir_path <- paste0(temp_folder, "/files/")
52
53 fs::dir_copy(dir_to_copy, new_dir_path, overwrite = TRUE)
54 fs::file_copy(output_kml, temp_folder, overwrite = TRUE)
55 file.rename(
56   from = file.path(temp_folder, kml_file_name),
57   to = file.path(temp_folder, "doc.kml")
58 )
59
60 #clean up all of the extra line breaks in the kml file
61 mystring <- readr::read_file(file.path(temp_folder, "doc.kml"))
62 mystring2 <- gsub('\r\r\r\r\r\n', '\n', mystring, fixed = T)
63 mystring3 <- gsub('\r\r\r\r\r\n', '\n', mystring2, fixed = T)
64 mystring4 <- gsub('\r\r\r\r\n', '\n', mystring3, fixed = T)
65 mystring5 <- gsub('\r\r\n', '\n', mystring4, fixed = T)
66 mystring6 <- gsub('\n\r\n', ' ', mystring5, fixed = T)
67
68 # Extract the part of the string after the last '/'
69
70 if (addOverlays == TRUE) {
71   last_part_dir <- tail(strsplit(dir_to_copy, "/")[[1]], 2)
72   mykml <-
73     stringr::str_replace_all(mystring6[1], paste0("src='./",
74       ↪ last_part_dir[1], "/", last_part_dir[2]), "src='files'")
75 } else if (addOverlays == FALSE) {
76   last_part_dir <- tail(strsplit(dir_to_copy, "/")[[1]], 2)
77   mykml <-
78     stringr::str_replace_all(mystring6[1], paste0("src='./",
79       ↪ last_part_dir[2]), "src='files'")
80 }
81
82 mykml <- stringr::str_replace_all(mykml[1], "<name>./", "<name>")
83 sink(paste0(file.path(temp_folder, "doc.kml")))
84 writeLines(mykml)
85 sink()
86
87 # name for new kmz file
88 kmz_file_name <-
89   paste0(usefun::get_parent_dir(directory), "/",
90     basename(tools::file_path_sans_ext(output_kml)),
91     ".kmz")

```

```

90
91 # create the kmz file
92 myWd <- temp_folder
93 files_lst <-
94   list.files(
95     path = temp_folder,
96     pattern = "*.jpg|*.kml",
97     all.files = FALSE,
98     full.names = FALSE,
99     recursive = TRUE,
100     ignore.case = FALSE,
101     include.dirs = FALSE
102   )
103
104 # zip the file up
105 zip::zip(
106   kmz_file_name,
107   files_lst,
108   recurse = FALSE,
109   compression_level = 9,
110   include_directories = TRUE,
111   root = myWd,
112   mode = "mirror"
113 )
114
115 # remove the temp folder and its contents
116 unlink(temp_folder, recursive = TRUE)

```