Package ‘uavRmp’

July 22, 2025

Type Package

Title UAV Mission Planner

Version 0.7

Date 2024-06-07

Encoding UTF-8

Maintainer Chris Reudenbach <reudenbach@uni-marburg.de>

Description
The Unmanned Aerial Vehicle Mission Planner provides an easy to use work flow for planning autonomous obstacle avoiding surveys of ready to fly unmanned aerial vehicles to retrieve aerial or spot related data. It creates either intermediate flight control files for the DJI-Litchi supported series or ready to upload control files for the pixhawk-based flight controller. Additionally it contains some useful tools for digitizing and data manipulation.

URL https://github.com/gisma/uavRmp

BugReports https://github.com/gisma/uavRmp/issues

License GPL (>= 3) | file LICENSE

Depends R (>= 3.1.0)

Imports sp, sf, geosphere, tools, log4r, zoo, methods, brew, exifr, link2GI, data.table, jsonlite, rlist, xfun, terra, concaveman, dplyr, spatialEco

RoxygenNote 7.3.1

SystemRequirements GNU make

Suggests knitr, rmarkdown, markdown, mapview, grDevices, stringr, htmltools, htmlwidgets,

VignetteBuilder knitr

NeedsCompilation no

Author Chris Reudenbach [cre, aut],
Marvin Ludwig [ctb],
Sebastian Richter [ctb],
Florian Detsch [ctb],
Hanna Meyer [ctb]

Repository CRAN

Date/Publication 2024-06-07 09:10:02 UTC

Contents

`makeAP` *UAV Mission Planning tool for autonomous monitoring flight tasks with respect to DSM/DEM, orthophoto data retrieval.*

Description

The basic idea is to provide an easy to use workflow for controlling rtf-UAVs for planning autonomous surveys to retrieve aerial data sets.

Usage

```
makeAP(
  projectDir = tempdir(),
  locationName = "flightArea",
  surveyArea = NULL,
  flightAltitude = 100,
  launchAltitude = NULL,
  followSurface = FALSE,
  followSurfaceRes = 25,
  demFn = NULL,
  noFiles = 1,
  altFilter = 1,
  horizonFilter = 30,
  flightPlanMode = "track",
  useMP = FALSE,
  presetFlightTask = "remote",
  overlap = 0.8,
  maxSpeed = 20,
  maxFlightTime = 10,
  picRate = 2,
  windCondition = 0,
  uavType = "pixhawk",
  cameraType = "MAPIR2",
  buf_mult = 1.5,
  cmd = 16,
  uavViewDir = 0,
  maxwaypoints = 9999,
  above_ground = FALSE,
  djiBasic = c(0, 0, 0, -90, 0),
  dA = FALSE,
  picFootprint = FALSE,
  rcRange = NULL,
  copy = FALSE,
  runDir = tempdir(),
  gdalLink = NULL
)
```

Arguments

`projectDir`	`character` path to the main folder where several locations can be hosted, default is `tempdir()`
`locationName`	`character` path to the location folder where all tasks of this plot are hosted, default is `"flightArea"`
`surveyArea`	you may provide either the coordinates by c(lon1,lat1,lon2,lat2,lon3,lat3,launchLat,launchLon) or an OGR compatible file (prefunable to find an inherited method for function 'makeAP' for signature '"missing"'erably geoJSON or KML) with at least 4 coordinates that describe the flight area. The fourth coordinate is the launch position. You will find further explanation under seealso.
`flightAltitude`	set the default flight altitude of the mission. It is assumed that the UAV is started at the highest point of the surveyArea otherwise you have to defined the position of launching.
`launchAltitude`	absolute altitude of launching position. It will overwrite the DEM based estimation if any other value than -9999
`followSurface`	`boolean` TRUE performs an altitude correction of the mission's flight altitude using additional DEM data. If no DEM data is provided and `followSurface` is TRUE, SRTM data will be downloaded and used. Further explanation at seealso
`followSurfaceRes`	horizontal step distance for analyzing the DEM altitudes
`demFn`	filename of the corresponding DEM data file.

`noFiles`	manual split number of files
`altFilter`	if `followSurface` is equal `TRUE` then `altFilter` is the threshold value of accepted altitude difference (m) between two way points. If this value is not exceeded, the way point is omitted due to the fact that only 99 way points per mission are allowed.
`horizonFilter`	integer filter size of the rolling filter kernel for the flight track. Must be multiplied by the `followSurfaceRes` to get the spatial extent
`flightPlanMode`	type of flight plan. Available are: `"waypoints"`, `"track"`, `"manual"`.
`useMP`	default is FALSE switches to use a missionplanner/Qgroundcontrolplanner survey as planning base
`presetFlightTask`	(DJI only) strongly recommended to use "remote" Options are: `"simple_ortho"` takes one picture/way point, `"multi_ortho"` takes 4 picture at a waypoint, two vertically down and two in forward and backward viewing direction and an angle of -60deg, `"simple_pano"` takes a 360 deg panorama picture and `"remote"` which assumes that the camera is controlled by the remote control (RC)
`overlap`	overlapping of the pictures in percent (1.0 = 100)
`maxSpeed`	cruising speed
`maxFlightTime`	user defined estimation of the lipo lifetime (20 min default)
`picRate`	fastest stable interval (s) for shooting pictures
`windCondition`	1= calm 2= light air 1-5km/h, 3= light breeze 6-11km/h, 4=gentle breeze 12-19km/h 5= moderate breeze 20-28km/h
`uavType`	type of UAV. currently "dji_csv" for Litchi CSV export and "pixhawk" for MAVlink compatible flightplans are supported
`cameraType`	depending on the UAV Platform and integrated camera choose for DJI Mini 1/2/3, Phantom 3/Phantom 4 , Inspire 1) the `dji43` and for the DJI Air 2S the `dji32` tag. For GoPro action cams on whatever aircraft you can choose `GP3_7MP` or `GP3_11MP`. Flying the Mapir 2 camera choose `MAPIR2`. For the E90X camera of Yuneec you choose `YUN90`. Please note the calculation of the flight pathes is done via the ratio of vertical and horizontal resolution of the camera in the NON 16:9 and Landscape Modus.
`buf_mult`	multiplier for defining the zone in which the waypoints are assumed to be turning waypoints according to buf_mult * `followSurfaceRes`
`cmd`	mavlink command
`uavViewDir`	view direction of uav
`maxwaypoints`	maximal number of waypoints for Litchi default is 90
`above_ground`	Litchi setting if the waypoint altitudes are interpreted as AGL default = FALSE
`djiBasic`	c(0,0,0,-90) curvesize (DJI only) controls the curve angle of the uav passing way points. By default it is set to (`= 0.0`). rotationdir (DJI only) camera control parameter set the UAV basic turn direction to right (0) or left (1)

	gimbalmode (DJI only) camera control parameter `0` deactivates the gimbal control `1` activates the gimbal for focusing POIs `2` activates the gimbal for focus and interpolate a field of view in an angel of `gimbalpitchangle` gimbalpitchangle (DJI only) vertical angle of camera `+30 deg..-90 deg` actiontype (DJI only) individual actionype settings of the camera c(1,1,...) actionparam (DJI only) corresponding parameter for the above individual actiontype c(0,0,...) `uavViewDir` viewing direction of camera default is `0`
`dA`	if TRUE the real extent of the used DEM is returned helpful for low altitudes flight planning
`picFootprint`	switch for calculating the footprint at all way points
`rcRange`	range of estimated range of remote control
`copy`	copy switch
`runDir`	`character` runtime folder
`gdalLink`	link to GDAL binaries

Details

makeAP (make aerial plan) creates either intermediate flight control files for the DJI phantom x UAVs or ready to upload control files for the 3DR Solo/PixHawk flight controller. The DJI control files are designed for using with the proprietary litchi flight control app exchange format, while the 3DR Solo/PixHawk flight controller files are using the MAVLINK common message set, that is used by the PixHawk flight controller family. Both are implemented very rudimentary.

DJI:
The reason using DJI is their absolute straightforward usage. Everybody can fly with a DJI but the price is a more or less closed system at least in the low budget segment. There are workarounds like the litchi app that provides additionally to a cloud based mission planner an offline/standalone interface to upload a CSV formatted way point file for autonomous flights to the Phantom.

PixHawk flight controller/3DR Solo:
The open UAV community is focused on the PixHawk autopilot unit and the Mission Planner software. It is well documented and several APIs are provided. Nevertheless a high resolution terrain following flight planning tool for autonomous obstacle avoiding flight missions is not available. `makeAP` creates a straightforward version of MAV format flight control rules that are ready to be uploaded directly on the Pixhawk controller using the `solo_upload` function.

Warning

Take care! There are still a lot of construction zones around. This script is far beyond to be in a mature state. Please control and backup all controls again while planning and performing autonomous flight plans and missions. You will have a lot of chances to make a small mistake what may yield in a damage of your UAV or even worse in involving people, animals or non-cash assets. Check your risk, use parachute systems and even if it is running like a charm, keep alert!

See Also

The underlying concept, a tutorial and a field guide can be found in the package vignettes. See `browseVignettes("uavRmp")` or `vignette(package = "uavRmp")` or at Github uavRmp manual).

Examples

```
## Not run:
# Depending on the arguments, the following spatial data sets can be returned:

# lp      the planned launching position of the UAV.
# wp      waypoints inclusive all information
# oDEM    the original (input) digital surface model (DSM)
# rDEM    the resampled (used) DSM
# fp      optimized footprints of the camera
# fA      flight area with at least 2 overlaps
# rcA     area covered by the RC according to the range and line of sight

## for visualisation and vecDraw load mapview
require(mapview)

## (1) get example DEM data
demFn <- system.file("extdata", "mrbiko.tif", package = "uavRmp")
tutorial_flightArea <- system.file("extdata", "flightarea.kml", package = "uavRmp")

## (2) simple flight, 100 meters above ground
##     assuming a flat topography,

fp <- makeAP(surveyArea = tutorial_flightArea,
              demFn = demFn)

## (3) typical real case scenario (1)
##     A flight altitudes BELOW 50 m is ambitious and risky
##     You have to use a high quality high resulution DSM
##     (here simulated with a standard DEM)

fp <- makeAP(surveyArea=tutorial_flightArea,
          followSurface = TRUE,
          flightAltitude = 45,
          demFn = demFn,
          windCondition = 1,
          uavType = "dji_csv",cameraType = "dji32",
          followSurfaceRes = 5,
          altFilter = .75)


## (4) typical real case scenario (2)
##     A flight altitudes BELOW 50 m is ambitious and risky
##     You have to use a high quality high resulution DSM
##     (here simulated with a standard DEM)
##    NOTE All settings are taken from QGroundcontrol so adapt the survey settings according
##        to "calc above terain" and use the "YUN90" camera tag for camera flight speed etc.
##     NOTE EXPERIMENTAL
```

```
demFn <- system.file("extdata", "mrbiko.tif", package = "uavRmp")
tutorial_flightArea <- system.file("extdata", "tutdata_qgc_survey.plan", package = "uavRmp")
fp <- makeAP(surveyArea=tutorial_flightArea,
            useMP = TRUE,
            followSurface = TRUE,
            demFn = demFn,
            windCondition = 1,
            uavType = "pixhawk",
            cameraType = "YUN90",
            followSurfaceRes = 5,
             altFilter = .75)

## (5) typical real case scenario (3)
##     This examples uses a flight planning from the QGroundcotrol Survey planning tool
##     It also used the all calculations for camera flight speed etc.
##     The flight plan is modyfied by splitting up the task according to 99 Waypoints
##     and flight time and saved as litchi csv format
##     NOTE EXPERIMENTAL tested with DJI mavic mini 2

demFn <- system.file("extdata", "mrbiko.tif", package = "uavRmp")
tutorial_flightArea <- system.file("extdata", "tutdata_qgc_survey.plan", package = "uavRmp")
fp <- makeAP(surveyArea=tutorial_flightArea,
            useMP = TRUE,
            demFn = demFn,
            maxFlightTime = 25,
            cameraType = "dji32",
            uavType = "dji_csv")

## call a simple shiny interface
shiny::runApp(system.file("shiny/plan2litchi/", "app.R", package = "uavRmp"))


## (6) view results

mapview::mapview(fp$wp,cex=4, lwd=0.5)+
mapview::mapview(fp$lp,color = "red", lwd=1,cex=4)+
mapview::mapview(fp$fA,color="blue", alpha.regions = 0.1,lwd=0.5)+
mapview::mapview(fp$oDEM,col=terrain.colors(256))



## (6) digitize flight area using the small "onboard" tool vecDraw()
##     save vectors as "kml" or "json" files
##     provide full filename  +  extension!


vecDraw(preset="uav")

## End(Not run)
```

makeTP *Flight Track Planning tool*

Description

makeTP generates a flight track chaining up point objects with respect to a heterogenous surface and known obstacles as documented by an DSM for taking top down pictures. It creates a single control file for autonomous picture retrieval flights.

Usage

```
makeTP(
  projectDir = tempdir(),
  locationName = "treePos",
  missionTrackList = NULL,
  launchPos = c(8.772055, 50.814689),
  demFn = NULL,
  flightAltitude = 100,
  climbDist = 7.5,
  aboveTreeAlt = 15,
  circleRadius = 1,
  takeOffAlt = 50,
  presetFlightTask = "remote",
  maxSpeed = 25,
  followSurfaceRes = 5,
  altFilter = 0.5,
  windCondition = 1,
  launchAltitude = -9999,
  uavType = "pixhawk",
  cameraType = "MAPIR2",
  copy = FALSE,
  runDir = ""
)
```

Arguments

`projectDir`	`character` path to the main folder where several projects can be hosted, default is `tempdir()`
`locationName`	`character` base name string of the mission, default is `"treePos"`
`missionTrackList`	`character` filename of the mission tracklist (target positions), default is `NULL`
`launchPos`	`list` launch position c(longitude,latitude), default is `c(8.772055,50.814689)`
`demFn`	`character` filename of the used DSM data file, default is `NULL`
`flightAltitude`	`numeric` set the AGL flight altitude (AGL while the provided raster model represents this surface) of the mission, default is `100` default is (= `0.0`). If set to `-99`

	it will be calculated from the swath width of the pictures. NOTE: This makes only sense for `followSurface = TRUE` to smooth curves. For `flightPlanMode = "waypoint"` camera actions (DJI only EXPERIMENTAL) are DISABLED during curve flights.
`climbDist`	`numeric` distance within the uav will climb on the caluclated save flight altitude in meter, default is `7.5`
`aboveTreeAlt`	`numeric` minimum flight height above target trees in meter, default is `15.0`
`circleRadius`	`numeric` radius to circle around above target trees in meter, default is `1.0`
`takeOffAlt`	altitude `numeric` climb altitude of the uav at take off position in meter, default is `50.0`
`presetFlightTask`	`character` (DJI only EXPERIMENTAL). NOTE: it is strongly recommended to use the default `"remote"` Further options are: `"simple_ortho"` takes one picture/waypoint, `"multi_ortho"` takes 4 picture at a waypoint, two vertically down and two in forward and backward viewing direction and an angele of -60deg, `"simple_pano"` takes a 360 deg panorama picture and `"remote"` which assumes that the camera is controlled by the remote control (RC)
`maxSpeed`	`numeric` cruising speed, default is `25.0`
`followSurfaceRes`	`numeric`, default is 5 meter.
`altFilter`	`numeric` allowed altitude differences bewteen two waypoints in meter, default is `0.5`
`windCondition`	`numeric`options are 1= calm 2= light air 1-5km/h, 3= light breeze 6-11km/h, 4=gentle breeze 12-19km/h 5= moderate breeze 20-28km/h, default is `1`
`launchAltitude`	`numeric` altitude of launch position. If set to `-9999` a DEM is required for extracting the MSL, default is `-9999`
`uavType`	`character` type of UAV. Currently "dji_csv" and "pixhawk" are supported, default is `"pixhawk"`
`cameraType`	`character`, default is `"MAPIR2"`.
`copy`	`boolean` copy used file to data folder default is `FALSE`
`runDir`	`character` runtime folder

Examples

```
## Not run:
## (1) get example DEM data
dsmFn <- system.file("extdata", "mrbiko.tif", package = "uavRmp")
## (2) make position flight plan
makeTP  <-  makeTP(missionTrackList= tutorial_flightArea,
                   demFn = dsmFn,
                   uavType = "pixhawk",
                   launchPos = c(8.679,50.856))

## End(Not run)
```

maxpos_on_line *applies a line to a raster and returns the position of the maximum value*

Description

applies a line to a raster and returns the position of the maximum value

Usage

```
maxpos_on_line(dem, line)
```

Arguments

dem	raster object
line	sp object

Examples

```
## Not run:
## load DEM/DSM
dem <- terra::rast(system.file("extdata", "mrbiko.tif", package = "uavRmp"))

## generate extraction line object
line <- sp_line(c(8.66821,8.68212),c(50.83939,50.83267),ID="Highest Position",runDir=runDir)
## extract highest position
maxpos_on_line(dem,line)

## End(Not run)
```

minBB *Rectangle flight area around points*

Description

Creates optimal rectangle area around points

Usage

```
minBB(points, buffer = 0, epsg = 25832)
```

Arguments

points	a sf object, points you want to fly over
buffer	buffer distance between the points and the rectangle; defaults 0
epsg	reference system

Details

The code is based on a Rotating Caliper Algorithm and mostly copy and pasted (see reference)

Value

SpatialPoints: Corners of the flight area

Author(s)

Marvin Ludwig

References

http://dwoll.de/rexrepos/posts/diagBounding.html

`soloLog` *Download, reorganize and export the binary log files from 3DR Solo Pixhawk controller or the telemetry log files from the Solo radio control unit*

Description

Wraps the mavtogpx.py converter as provided by the dronkit library). It downloads and optionally converts the most important 3DR Solo logfiles. Optionally you may import the geometries and data as `sp` object.

Usage

```
soloLog(
  logFileSample = "recent",
  logSource = "rc",
  logDest = tempdir(),
  downloadOnly = FALSE,
  netWarn = FALSE,
  renameFiles = TRUE,
  makeSP = FALSE
)
```

Arguments

`logFileSample`	`character` , options are: `recent` download the most recent logfile, `all` downloads all logfiles, or a plain number e.g. `2` for a specific logfile. Note the telemetry logfiles are numbering from 1 to 9 only, the most recent one is not numbered. The binary logfiles from the pixhawk are numbering continously but only the last 50 files or so will exist.

logSource	character, options are: rc = logfiles from the radio control, pixhawk = logfiles from the flightcontroller, default is set to rc. The radio control is providing the last ten telemetry data files, while the flight controller provides the latest 50 binary logfiles.
logDest	character (existing) destination path to which the logs should be downloaded to
downloadOnly	logical wether to only download the files or also convert and rename them, default is set FALSE
netWarn	logical wether to warn and waits before starting a connection to the controller. helps while testing due to occassional wifi shutdowns of the Solo, default is set to FALSE
renameFiles	logical renames the log and gpx files according to the time period, default is set TRUE
makeSP	logical wether returning an sp object from the gpx files or not, default is FALSE

Note

for using the Solo stuff is tested only for Linux and the bash shell under Windows 10. You need to install the following python libs:

```
sudo pip install pymavlink
sudo pip install dronekit-sitl
sudo pip install dronekit
```

Additionally you need sshpass:

```
sudo apt-get install sshpass
```

And please rememeber - you need to be connected at least to a running 3DR Solo radio control and if you want to donload data from the Pixhawk to a Solo UAV

Examples

```
## Not run:
## download recent telemetry log file from controller and convert it to gpx
soloLog(logFiles = "solo.tlog")

## download the last available logfile from the radio control
soloLog()

## download ALL logfiles from the radio control
soloLog(logFiles = "all")

## download ALL telemetry logfiles from the flight controller
soloLog(logSource = "pixhawk",logFiles = "all")

## download telementry logfile number 5  from the remote control
soloLog(logSource = "rc",logFiles = "5")

## End(Not run)
```

solo_upload — *Upload MAV compliant mission File to a 3DR Solo*

Description

solo_upload provides a crude interface to upload the Solo mission file to the 3dr SOLO

Usage

```
solo_upload(
  missionFile = NULL,
  connection = "udp:10.1.1.166:14550",
  prearm = "-1"
)
```

Arguments

`missionFile`	mission file to upload
`connection`	a valid connection string to the Solo default is "udp:10.1.1.166:14550"
`prearm`	`character` controls the prearm status of the Solo prearm check 0=Disabled 1=Enabled -3=Skip Baro -5=Skip Compass -9=Skip GPS -17=Skip INS -33=Skip Params/Rangefinder -65=Skip RC 127=Skip Voltage default is `-1` Find more information at prearm safety, Mission import export script.

Note

Becareful with fooling around with the prearm stuff. It is kind of VERY sensitive for the later autonomous flights!
For using the Solo stuff you need to install:
sudo pip install pymavlink;
sudo pip install dronekit-sitl;
sudo pip install dronekit;
sudo apt-get install sshpass
Additionally you need to be connected to a running 3DR Solo uav

Examples

```
wp <- system.file("extdata", "MAVLINK_waypoints.txt", package = "uavRmp")
## Not run:
solo_upload( missionFile = wp)

## End(Not run)
```

---

sp_line *create an spatiallineobject from 2 points*

Description

create an spatiallineobject from 2 points, optional export as shapefile

Usage

```
sp_line(
  Y_coords,
  X_coords,
  ID = "ID",
  proj4 = "+proj=longlat +datum=WGS84 +no_defs",
  export = FALSE,
  runDir
)
```

Arguments

`Y_coords`	Y/lat coordinates
`X_coords`	X/lon coordinates
`ID`	id of line
`proj4`	projection
`export`	write shafefile default = F
`runDir`	`character` runtime folder

Examples

```
## Not run:
## creating sp spatial point object
line <- sp_line(c(8.770367,8.771161,8.771536),
                c(50.815172,50.814743,50.814875),
                runDir=tempdir())

## plot it
plot(line)

## End(Not run)
```

sp_point *create an spatialpointobject from 1 point*

Description

create an spatial point object from 1 point and optionally export it as a shapefile

Usage

```
sp_point(
  lon,
  lat,
  ID = "point",
  proj4 = "+proj=longlat +datum=WGS84 +no_defs",
  export = FALSE,
  runDir = runDir
)
```

Arguments

`lon`	lon
`lat`	lat
`ID`	name of point
`proj4`	projection
`export`	write shafefile default = F
`runDir`	`character` runtime folder

Examples

```
## creating sp spatial point object
point <- sp_point(8.770362,50.815240,ID="Faculty of Geographie Marburg")
```

tutdata_dem *DEM data set of Marburg-Biedenkopf*

Description

DEM data set resampled to 20 m resolution

Format

`"terra::rast"`

Details

DEM data set of Marburg-Biedenkopf

Source

`Faculty of Geography UAV derived data from Marburg University Forest first campaign`

`tutdata_dji` *DJI image of a survey flight*

Description

DJI image of a survey flight

Format

`"terra::rast"`

Details

DJI image of a survey flight

Source

`Faculty of Geography UAV derived data from Marburg University Forest first campaign`

`tutdata_flightarea` *Flight area planning example data*

Description

Flight area planning example data as typically needed for planning an autonomous survey flight task

Details

Flight area planning example data

Source

`Faculty of Geography Marburg`

`tutdata_flighttrack` *GPX example data*

Description

GPX example data as derived by a 3DR Solo flight

Details

GPX example data

Source

`Faculty of Geography UAV derived data from Marburg University Forest first campaign`

`tutdata_position` *position example data*

Description

position data for planning a single flight task with focus on known objects

Details

Virtual object position coordinates example data

Source

`Faculty of Geography UAV derived data from Marburg University Forest first campaign`

`tutdata_qgc_survey` *Flight area planning Qgroundcontrol planning file for a 100m relative to launch survey flight using a GoPro Hero4*

Description

Flight area planning example data as typically needed for planning an autonomous survey flight task. The task is planned with the QGroundcontrol survey tool.

Details

Flight area planning Qgroundcontrol survey data 100 m AGL

Source

`Faculty of Geography Marburg`

`tutdata_qgc_survey30m` *Flight area planning Qgroundcontrol planning file for a 30m follow terrain survey flight with the DJI Air 2S*

Description

Flight area planning example data as typically needed for planning an autonomous survey flight task. The task is planned with the QGroundcontrol survey tool.

Details

Flight area planning Qgroundcontrol survey data 30 m AGL

Source

`Faculty of Geography Marburg`

`tutdata_waypoints` *MAVLINK waypoint example data*

Description

Waypoint file

Details

MAVLINK waypoint example data

Source

`Faculty of Geography UAV derived data from Marburg University Forest first campaign`

`vecDraw` *digitizing vector features using a simple leaflet base map*

Description

vecDraw is designed for straightforward digitizing of simple geometries without adding attributes. It provides a bunch of leaflet base maps and optionally a sf* object can be loaded for orientation.

Usage

```
vecDraw(
  mapCenter = NULL,
  zoom = 15,
  line = TRUE,
  rectangle = TRUE,
  poly = TRUE,
  circle = TRUE,
  point = TRUE,
  remove = TRUE,
  position = "topright",
  maplayer = c("CartoDB.Positron", "OpenStreetMap", "Esri.WorldImagery",
    "Thunderforest.Landscape", "OpenTopoMap"),
  overlay = NULL,
  preset = "all",
  locPreset = "muf",
  cex = 10,
  lwd = 2,
  opacity = 0.7
)
```

Arguments

`mapCenter`	center of the leaflet map
`zoom`	set initial zoom level of leaflet map
`line`	enable/disable line tool
`rectangle`	enable/disable polygon tool
`poly`	enable/disable polygon tool
`circle`	enable/disable circle tool
`point`	enable/disable point tool
`remove`	enable/disable the remove feature of the draw tool
`position`	toolbar layout (topright, topleft, bottomright, bottomleft)
`maplayer`	string as provided by leaflet-provider
`overlay`	optional sp* object may used for orientation
`preset`	character default is "uav" for line based mission digitizing, "ext" for rectangles, NULL for all drawing items
`locPreset`	character location preset, default is "muf" for Marburg University Forest, "tra" Traddelstein, "hag" Hagenstein, "baw" Bayerwald.
`cex`	size of item
`lwd`	line width of item
`opacity`	opacity of item

Note

Yu can either save the digitized object to a json (JS) or kml (KML) file.

Examples

```
## Not run:
# fully featured without overlay
require(mapview)

# preset for digitizing uav flight areas using Meuse data set as overlay
require(sp)
data(meuse)
sp::coordinates(meuse) <- ~x+y
sp::proj4string(meuse) <-CRS("+init=epsg:28992")
m <- sp::spTransform(meuse,CRSobj = sp::CRS("+init=epsg:4326"))
vecDraw(overlay = m, preset = "uav")

# preset for digitizing simple rectangles extents
vecDraw(preset="ext",overlay = m)

## End(Not run)
```

Index