Package ‘uavRmp’

October 28, 2018

Type Package

Title UAV Mission Planner

Version 0.5.4

Date 2018-10-20

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 (almost) ready to fly unmanned aerial vehicles to retrieve aerial or spot related data. It creates either intermediate flight control files for the DJI phantom series or ready to upload control files for the pixhawk based flight controller as used in the 3DR Solo. Additionally it contains some useful tools for digitizing and data manipulation.

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

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

LazyData TRUE

License GPL (>= 3) | file LICENSE

Depends R (>= 3.1.0)

Imports stringr, sp, raster, htmlwidgets, htmltools, rgdal, rgeos, gdalUtils, geosphere, tools, maptools, log4r, zoo, data.table, spatial.tools, devtools, sf, roxygen2, methods, brew

RoxygenNote 6.1.0

SystemRequirements GNU make

Suggests knitr, rmarkdown, mapview

VignetteBuilder knitr

NeedsCompilation no

Author Chris Reudenbach [cre, aut],
    Sebastian Richter [ctb],
    Lars Oppenooth [ctb],
    Florian Detsch [ctb],
    Hanna Meyer [ctb]
Description

copyDir copy all image data to the corresponding folder

Usage

copyDir(fromDir, toProjDir, pattern = "*")

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fromDir</td>
<td>character a path to the image data</td>
</tr>
<tr>
<td>toProjDir</td>
<td>character a path to the projRootDir</td>
</tr>
<tr>
<td>pattern</td>
<td>character a string pattern for filtering file list</td>
</tr>
</tbody>
</table>
initProj

Defines and creates folders and variables

Description

Defines and creates (if necessary) all folders variables set the SAGA path variables and other system variables exports all variables to the global environment

Usage

initProj(projRootDir = getwd(), projFolders = c("log/", "control/", "run/", "data/"))

Arguments

projRootDir       project github root directory (your github name)
projFolders      list of subfolders in project

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 = NULL, demFn = NULL, altFilter = 1, horizonFilter = 30, flightPlanMode = "track", presetFlightTask = "remote", overlap = 0.8, maxSpeed = 20, maxFlightTime = 10, picRate = 2, windCondition = 0, uavType = "solo", cameraType = "MAPIR2", cmd = 16, uavViewDir = 0, djiBasic = c(0, 0, 0, -90, 0), dA = FALSE, heatmap = FALSE, picFootprint = FALSE, rcRange = NULL, copy = FALSE, runDir = tempdir())
**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 (preferably to find an inherited method for function `makeAP` for signature "missing" easily 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.
- **altFilter** if `followingTerrain` 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".
- **presetFlightTask** (DJI only) strongly recommended to use "remote" Options are: "simple_ortho" takes one picture/way point, "multi_ortho" takes 4 picture at a wauable to find an inherited method for function `makeAP` for signature "missing" unable to find an inherited method for function `makeAP` for signature "missing" ypoint, 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)
- **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
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. The DJI control files are designed for using with the proprietary litchi flight control app exchange format, while the 3DR Solo files are using the MAVLINK common message set, that is used by the PixHawk flight controller family. Both are implemented very rudimentarily.

DJI:
The reason using DJI is their absolute straightforward usage. Everybody can fly with a DJI but the price is a hermetically closed system. Only the litchi app provides additionally to a cloud based mission planner an offline/standalone interface to upload a CSV formated way point file for autonomous flights to the Phantom.

PixHawk/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!

Note

To use the script you need to install quite a lot of R-packages and at least the binary GDAL tools as well as SAGA GIS and GRASS GIS according to your system needs. Please find more information at the giswerk.org: "uav based Remote Sensing at giswerk.org). https://gisma.github.io/

See Also

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

Examples

```r
## Not run:
# Depending on the arguments, the following spatial data sets can 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
# hm a heatmap abundance of pictures/pixel (VERY SLOW, only if heatMap = TRUE)

## 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
## Flight altitudes BELOW 50 m is ambitious and risky
## You have to use a high quality high resolution DSM
## (here simulated with a standard DEM)
```
fp <- makeAP(surveyArea=tutorial_flightArea,
    followSurface = TRUE,
    flightAltitude = 45,
    demFn = demFn,
    windCondition = 1,
    uavType = "solo",
    followSurfaceRes = 5,
    altFilter = .75)

## (4) 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))

## (5) 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)

---

**makeGlobalVar**

*Generates a variable with a certain value in the R environment*

### Description

Generates a variable with a certain value in the R environment

### Usage

```r
makeGlobalVar(name, value)
```

### Arguments

- `name` character string name of the variable
- `value` character string value of the variable

### Examples

```r
## Not run:

# creates the global var `pathToData` with the value `~/home/data`
makeGlobalVar("pathToData", "/home/data")

## 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 a 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 = "solo", 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 followingTerrain = 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 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)

maxSpeed numeric cruising speed, default is 25.0
followSurfaceRes numeric, default is 5 meter.
altdFilter numeric allowed altitude differences between 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 "djip3" and "solo" are supported, default is
"solo"
cameraType character, default is "MAPIR2".
copy boolean copy used file to data folder default is FALSE
runDir character runtime folder

Examples

```r
## 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 = "solo",
  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)`
Read GPX file

Description

Read a GPX file. By default, it reads all possible GPX layers, and only returns shapes for layers that have any features.

Usage

read_gpx(file, layers = c("waypoints", "tracks", "routes", "track_points", "route_points"))

Arguments

file a GPX filename (including directory)
layers vector of GPX layers. Possible options are "waypoints", "tracks", "routes", "track_points", "route_points". By default, all those layers are read.

Value

if the layer has any features a sp object is returned.

Note
cloned from tmap
Examples

```r
## for visualisation we are using mapview
require(mapview)
## assign GPX file
gpxFN <- system.file("extdata", "flighttrack.gpx", package = "uavRmp")

## read it
gpx <- read_gpx(gpxFN, layers=c("tracks"))

## plot it
mapview::mapview(gpx)
```

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

```r
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 whether to only download the files or also convert and rename them, default is set FALSE
solo_upload

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

```bash
## 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
**solo_upload**

**Usage**

```python
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**

Be careful 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**

```r
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 = NULL, 
proj4 = "+proj=longlat +datum=WGS84 +no_def", export = FALSE, 
rundir = NULL)

arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>y_coords</td>
<td>Y/lat coordinates</td>
</tr>
<tr>
<td>x_coords</td>
<td>X/lon coordinates</td>
</tr>
<tr>
<td>id</td>
<td>id of line</td>
</tr>
<tr>
<td>proj4</td>
<td>projection</td>
</tr>
<tr>
<td>export</td>
<td>write shapefile default = F</td>
</tr>
<tr>
<td>rundir</td>
<td>character runtime folder</td>
</tr>
</tbody>
</table>

examples
## creating sp spatial point object
line <- sp_line(c(8.770367, 8.771161, 8.771536), 
c(50.815172, 50.814743, 50.814875), 
ID = "go for it", 
rundir = rundir)

## plot it
raster::plot(line)

sp_point

create an spatialpointobject from 1 points

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_def", export = FALSE, 
rundir = rundir)
tutdata_dem

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

Examples

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

## plot it
raster::plot(tutdata_dem)
```

Description

DEM data set resampled to 20 m resolution

Format

"raster::raster"

Details

DEM data set of Marburg-Biedenkopf

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
### Description

Waypoint file

### Details

MAVLINK waypoint example data

### Source

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

---

### Description

The package provides some mission planning functionality for dealing with Unmanned Aerial Vehicles. The focus is set on an easy to use workflow for planning autonomous obstacle avoiding surveys of rtf-UAVs to retrieve aerial or spot related data. It provides either intermediate flight control files for the DJI phantom x UAVs or ready to upload control files for the pixhawk based flightcontroller as used in the 3DR Solo. Additionally it contains some useful tools for digitizing and data manipulation.

### Details

The package provides some mission planning functionality for dealing with Unmanned Aerial Vehicles

### Note

It is important to keep in mind that all auxilliary external binaries like GDAL or SAGA need to be installed properly. correctly on your system.

### Author(s)

Chris Reudenbach Lars Opgenoorth Sebastian Richter Florian Detsch Hanna Meyer

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

vecdrawHmapcenter = nullL zoom = 1UL line = trueL rectangle = trueL
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

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

# 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)
Index

*Topic **datasets**
  * tutdata_dem, 15
  * tutdata_flightarea, 16
  * tutdata_flighttrack, 16
  * tutdata_position, 16
  * tutdata_waypoints, 17

*Topic **package**
  * uavrmp, 17

  copyDir, 2

  initProj, 3

  makeAP, 3
  makeGlobalVar, 7
  makeTP, 8
  maxpos_on_line, 9

  read_gpx, 10

  solo_upload, 12
  soloLog, 11
  sp_line, 14
  sp_point, 14

  tutdata_dem, 15
  tutdata_flightarea, 16
  tutdata_flighttrack, 16
  tutdata_position, 16
  tutdata_waypoints, 17

  uavrmp, 17
  uavrmp-package (uavrmp), 17

  vecDraw, 18