

Documentation MR1Analysis Tool



Project:	MR1Analysis Tool
Contractor:	Swiss Birdradar Solution AG
Author:	Fabian Hertner
Project Leader:	Dominik Kleger
Version:	1.0
Date:	08.04.21

Table of Content

1 Introduction	3
2 MR1 Data	4
2.1 Database structure	4
2.2 Database Tables	6
2.2.1 Collection	6
2.2.2 time_bins	6
2.2.3 protocol	7
2.2.4 visibility	7
2.2.5 site	8
2.2.6 radar	9
2.2.7 weather and weather_property	9
2.2.8 echo_rfeatures_map and rfeatures	10
2.2.9 echo_validation and echo_validation_type	10
2.2.10 rf_classification and rfclasses	11
2.2.11 rf_class_probability	11
2.3 RF Features	13
2.3.1 Statistical Features	13
2.3.2 Overall Shape Features	13
2.3.3 STFT Features	14
2.3.4 Auto Correlation Features	15
2.3.5 Wing Flapping Packages	15
2.3.6 RCS Features	17
2.3.7 Low Frequency Features	17
2.3.8 Polarization / Shape Features	17
2.3.9 Wing Beat Frequency	18
3 MR1Analysis Tool	19
3.1 Installation	19
3.2 Analyze Data	19
3.2.1 Extract Data from Database	20
3.2.2 Sunrise/Sunset	21
3.2.3 Add Day/Night Information to Echo Data	21
3.2.4 Filter Protocol Data	22
3.2.5 Blind Times	23
3.2.6 Filter Echo Data	24
3.2.7 Altitude Bins	25
3.2.8 Time Bins	26
3.2.9 Observation Times	26
3.2.10 MTR Computation	28
3.2.11 Plot MTR	30

1 Introduction

This document describes the use of the data produced by the BirdScan MR1 v1.6.0.7 and greater as well as the functionality of the MR1Analysis Tool. The MR1Analysis Tool is a collection of functions, written in “R”, to extract the data from the database and perform basic analysis such as MTR computation, wing-beat and direction analysis.

The MR1 Analysis Tool requires data collected by a BirdScan MR1 running BirdScan software version 1.6 or higher. Data collected by older BirdScan versions, reclassified with BirdScan v1.6 or higher is supported as well.

With the BirdScan v1.6 a new classifier based on the random forest algorithm was released. The previous naive bayes based classification is still computed and stored in the database as well as all the features.

2 MR1 Data

2.1 Database structure

The structure of the database (BirdScan v1.6) is shown in Figure 1. The figure shows the tables used by the MR1 Analysis tool. The database contains more tables than shown in Figure 1 which are not relevant for analysis purposes.



2.2 Database Tables

2.2.1 Collection

The collection table stores information about the detected echoes, one row per echo.

key type	name	datatype	linked to table
primary key	row	int	-
foreign key	protocolID	int	protocol
foreign key	time_bin	int	time_bins

Table 1: collection table keys

column	datatype	description	outdated by birdscan v1.6
echoID	int	Echo ID	no
protocolID	int	link to protocol table	no
stc_level	float	stc_level in dBm, used to calculate mtr-factors	no
mtr_fact	float	mtr-factor based on the old classification	yes
time_bin	int	link to time_bins table	no
statistical_classification	smallint	naive bayes classification	yes
time_stamp	datetime	timestamp of the echo	no
feature1	float	altitude in meters a.g.l.	no
feature2	float	azimuth in degree	no
feature3	float	speed in m/s	no
feature6	float	radar rotation frequency in Hz	no
feature14	float	maxlevel received in dBm	no
feature15	float	polarisation ratio	yes
feature16	float	absolute polarisation	yes
feature17	float	radar cross section (RCS) in m ²	no
feature18	float	square root of RCS	no
feature19	float	duration of echo in s	no
feature20	float	duration of echo in STC	no
feature24	float	alpha: angle between entry- and exit angle in the detection range in degrees	no

Table 2: important columns in collection table

2.2.2 time_bins

Every 5 minutes a new time bin is created and stored in the time_bins table. The time_bins table does not contain any information but start and stop time of the time bin. Other tables are linked to the time_bins table and store information per time bin or allow filtering by time bin.

key type	name	datatype	linked to table
primary key	id	int	-
foreign key	siteID	int	site

Table 3: time_bins table keys

column	datatype	description	outdated by birdscan v1.6
time_start	datetime	start time and date of the time bin	no
time_stop	datetime	stop time and date of the time bin	no

Table 4: important columns in time_bins table

2.2.3 protocol

The protocol table contains information about the operation mode of the radar. On each change of the operation mode (pulse-type, rotation, ...) a new protocol is added in the protocol table.

key type	name	datatype	linked to table
primary key	protocolID	int	-
foreign key	siteID	int	site

Table 5: protocol table keys

column	datatype	description	outdated by birdscan v1.6
protocolID	int	protocol ID	no
siteID	int	site ID	no
startTime	datetime	start time and date of the protocol	no
stop_time	datetime	stop time and date of the protocol	no
pulseType	varchar	pulsetype: short, medium, long ("S","M","L")	no
rotate	bit	radar in rotation mode (1/0)	no
stc	float	STC altitude in m	no
threshold	float	stc threshold in dBm	no
blockTime	float	time radar is blind after protocol change in secs	no

Table 6: important columns in protocol table

2.2.4 visibility

The visibility table store the times when the radar was blind due to weather conditions (rain, snow, heavy fog), clutter (e.g.: radome covered with snow or leaves, too many objects like trees, houses or vehicles nearby) or protocol changes. Blindtimes in the visibility table are linked to one protocol and are thus split if expanding over two or more protocols.

key type	name	datatype	linked to table
primary key	visibilityLogID	int	-
foreign key	protocolID	int	protocol

Table 7: visibility table keys

column	datatype	description	outdated by birdscan v1.6
visibilityLogID	int	visibility ID	no
protocolID	int	link to protocol table	no
blind_from	datetime	start time from when the radar was blind	no
blind_to	datetime	stop time to when the radar was blind	no

Table 8: important columns in visibility table

2.2.5 site

The site table contains information about the site where the radar is placed. One row per site.

key type	name	datatype	linked to table
primary key	siteID	int	-
foreign key	radarID	int	radar

Table 9: site table keys

column	datatype	description	outdated by birdscan v1.6
siteID	int	site ID	no
siteCode	varchar	3-letters abbreviation of the sitename	no
radarID	smallint	radar ID	no
siteName	varchar	name of the site	no
siteDesc	varchar	description of the site	no
projectStart	date	startdate of the project/campaign	no
projectEnd	date	enddate of the project/campaign	no
longitude	float	longitude of the site location in decimal degrees	no
latitude	float	latitude of the site location in decimal degrees	no
altitude	smallint	altitude a.s.l. of the site location in m	no
timeShift	varchar	time shift/zone at the site location	no
radarOrientation	float	orientation of the radar to north on site	no

Table 10: important columns in site table

2.2.6 radar

The radar table contains information about the radar hardware. For general analysis these values are not of interest.

key type	name	datatype	linked to table
primary key	radarID	int	-

Table 11: radar table keys

column	datatype	description	outdated by birdscan v1.6
radarID	smallint	radar ID	no
type	varchar	radar Type	no
serialNo	smallint	serial number	no

Table 12: important columns in radar table

2.2.7 weather and weather_property

The weather table contains information about the environment. The temperature/humidity sensors are placed inside the radar (one inside the radome and one behind the ventilation inlet) and do not represent the outside air temperature and humidity. The weather table is linked to the time_bins table and the weather_property table. One row per time bin and weather property is created.

The weather_property table describes all weather properties.

key type	name	datatype	linked to table
primary/foreign key	time_bin	int	time_bins
primary/foreign key	weather_property	int	weather_property

Table 13: weather table keys

key type	name	datatype	linked to table
primary key	id	int	-

Table 14: weather_property table keys

property	id	unit
Temperature Radome	1	degree C
Temperature Fresh Air	2	degree C
Relative Humidity Radome	3	%
Relative Humidity Fresh Air	4	%
Blind Time Percent	5	%

Table 15: weather properties

2.2.8 echo_rfeatures_map and rfeatures

The rfeatures table lists all features that are computed by the algorithms. A selection of them is used by the random forest models for classification and wing beat frequency estimation. The wing beat frequency and its credibility are part of the rfeatures and thus saved in the echo_rfeatures_map table. The echo_rfeatures_map contains one row per echo and rfeature.

The features are roughly described in chapter 2.3.

key type	name	datatype	linked to table
primary/foreign key	echo	int	collection
primary/foreign key	feature	int	rfeatures

Table 16: echo_rfeatures_map table keys

key type	Name	datatype	linked to table
primary key	Id	int	-

Table 17: rfeatures table keys

2.2.9 echo_validation and echo_validation_type

The echo validation algorithm of the BirdScan software validates each echo, if it is a valid bio scatterer or non-bio scatterer like rain, snow or ground clutter. The two types are listed in the echo_validation_type table and in the echo_validation table one row per echo is listed with its validation.

The echo validation algorithm is only working with raw radar data; therefore, the echo validation is not available for reclassified databases created by BirdScan versions older than v1.6.

key type	name	datatype	linked to table
primary/foreign key	echo_id	int	collection
foreign key	type	int	echo_validation_type

Table 18: echo_validation table keys

key type	name	datatype	linked to table
primary key	id	int	-

Table 19: echo_validation_type table keys

property	id
bio scatterer	1
non-bio scatterer	2

Table 20: echo validation types

2.2.10 rf_classification and rfclasses

the rf_classification table contains the classification based on the random forest classifier released with BirdScan version v1.6. The available classes are listed in the rfclasses table. The rfclasses table lists more classes than used by the classifier. Refer to the column 'isUsedForClassification' to get the classes used by the classifier.

key type	name	datatype	linked to table
primary/foreign key	echo	int	collection
foreign key	class	int	rfclasses

Table 21: rf_classification table keys

key type	name	datatype	linked to table
primary key	id	int	-

Table 22: rfclasses table keys

column	datatype	description	outdated by birdscan v1.6
echo	int	echo ID, link to collection table	no
class	int	class ID, link to rfclasses table	no
mtr_factor	real	MTR factor based on the random forest classifier	no
class_probability	real	classification probability	no

Table 23: important columns in rf_classification table

2.2.11 rf_class_probability

The rf_class_probability table is like the rf_classification table linked to the collection and rfclasses table. It lists the probabilities as well as the MTR factors for all classes. The MTR factor is dependent

on the size of the object (RCS). For small RCS, the RCS is replaced by a typical size for the class. Therefore, the MTR factor for one echo maybe different according to the classification.

This table exists for the case if one wants to change classifications manually, the probabilities and MTR factors do not have to be recalculated.

The table contains one row per echo and class.

key type	name	datatype	linked to table
primary/foreign key	echo	int	collection
primary/foreign key	class	int	rfclasses

Table 24: *rf_class_probability* table keys

2.3 RF Features

The radar computes a set of features which are used for the random forest models of the wing beat frequency estimator and the classifier. Both the wing beat frequency and the classifier use a subset of the features.

2.3.1 Statistical Features

The following statistical values are computed both for the raw and bandpass signal:

- Sample standard deviation
- Skewness
- Kurtosis

Additionally, the duration of the signal is saved.

#	Feature	DB-Name	ID
1	duration of the signature	duration_sec	0
2	standard deviation of the raw signal	std_sigRaw2	1
3	skewness of the raw signal	skw_sigRaw2	2
4	kurtosis of the raw signal	krt_sigRaw2	3
5	standard deviation of the band-passed signal	std_sigBP	4
6	skewness of the band-passed signal	skw_sigBP	5
	kurtosis of the band-passed signal	krt_sigBP	6

Table 25: basic features

2.3.2 Overall Shape Features

The overall shape features represent the changes respectively the consistency of the signal over time.

The signal is divided into time chunks. Within each chunk, the mean of the raw signal and the standard deviation of the bandpass signal is computed. Afterwards, the mean and standard deviation differences from one chunk to another are computed. The minimum, maximum and mean value of the differences are used as features.

#	Feature	DB-Name	ID
1	min difference of mean	shapeFeatures_1	7
2	mean difference of mean	shapeFeatures_2	8
3	max difference of mean	shapeFeatures_3	9
4	min difference of std	shapeFeatures_4	10
5	mean difference of std	shapeFeatures_5	11
6	max difference of std	shapeFeatures_6	12

Table 26: overall shape features

2.3.3 STFT Features

The STFT (Short Time Fourier Transform) features are based on the frequency spectrum of the high-pass power signal (in watt), calculated via the STFT. The different features are:

- The simple features are the entropy of the spectrum, the frequency at the maximum of the spectrum as well as the moments.
- The frequency at the maximum peak of the harmonic product spectrum and its entropy is computed for different numbers of down sampling factors.
- The peak detection computes different values of the peaks in the spectrum, such as the frequency of the highest peak, the relative height of it, the uniqueness of the highest peak and the inverted variance of the frequency differences of all detected peaks.
- The broad bin spectrum are the mean values of bins over the spectrum.

#	Feature	DB-Name	ID
1	entropy of the spectrum	spec_entropy	13
2	frequency of the maximum value of the spectrum	spec_mod_freq	14
3	mean of the spectrum	spec_mea_freq	15
4	standard deviation of the spectrum	spec_std_freq	16
5	skewness of the spectrum	spec_skw_freq	17
6	kurtosis of the spectrum	spec_krt_freq	18
7	frequency of the highest peak in the spectrum	StftFreqOfHighestPeak	19
8	relative height of the highest peak in the spectrum	StftRelHeightOfHighestPeak	20
	uniqueness of the highest peak in the spectrum	StftUniquenessOfHighestPeak	21
	inverted variance of frequency differences between the peaks	StftInvVarOfPeakDiffs	22

	fundamental frequency of harmonic product spectrum	HPSfundFreHz1 - 10	23 - 32
	entropy of harmonic product spectrum	HPSentropy1 - 10	33 - 42
	Broad Bins	BroadSpec_1 - 30	43 - 72

Table 27: STFT features

2.3.4 Auto Correlation Features

The autocorrelation features are based on the high-pass power (in watt) signal. The following features are computed:

- A sine-wave regression (least square) is processed on the autocorrelation. The sine wave frequency of the best fit and its regression error are used as features.
- A peak detection detects the highest peak, first peak and first peak above zero. The frequency and height of the peaks are used as features.

Structure in the Features class

#	Feature	DB-Name	ID
1	sine wave regression frequency	ACSineRegrFreq	73
2	sine wave regression error	ACSineRegrError	74
3	frequency of first peak	ACFirstPeakFreq	75
4	height of first peak	ACFirstPeakHeight	76
5	frequency of highest peak	ACMaxPeakFreq	77
6	height of highest peak	ACMaxPeakHeight	78
7	frequency of first peak above zero	ACFirstPeakAboveZeroFreq	79
8	height of first peak above zero	ACFirstPeakAboveZeroHeight	80

Table 28: auto correlation features

2.3.5 Wing Flapping Packages

To detect the wing flapping packages (wfp), the bandpass signal is used as base. The band-pass signal is converted to absolute values. To compute the wfp, several filters are applied:

- Moving mean of the absolute bandpass signal with a narrow window (0.15s)
- Moving maximum of the moving mean with a wide window (2.25s) for the upper envelope
- Moving minimum of the moving mean with a wide window (2.25s) for the lower envelope
- Moving ascending/descending test of the moving mean to indicate if the moving mean is varying or only ascending or descending

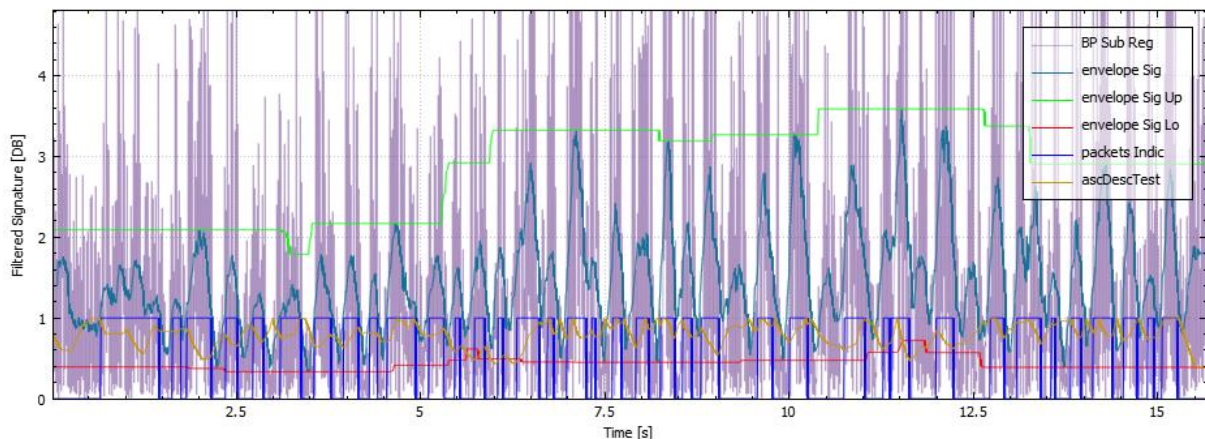


Figure 2: wing flapping packages

To detect the flapping and pause phases, different values are compared to thresholds:

- Ratio between lower and upper envelope
- Ratio between upper envelope and absolute maximum of upper envelope
- Moving mean (compared to a local threshold)
- Ascending/descending test

If those four conditions are true, a sample of the signal is considered as flapping, otherwise as pause.

This creates a binary signal and based on this, the wfp features are computed. These are:

- Proportion of flapping duration per signal duration
- Number of flapping packets, mean duration and standard deviation
- Number of pause packets, mean duration and standard deviation

Additionally, the spectrum of moving mean of the absolute bandpass signal (0.15s window) is computed via the STFT, and the lowest 28 frequency bins are used as features (ampiSpecFeat).

Structure in the Features class:

#	Feature	DB-Name	ID
1	proportional feature phase	WFPf_proportionFeat_phase	113
2	number of packets phase	WFPf_numP_phase	114
3	mean duration phase packets	WFPf_meadur_phase	115
4	duration of packets phase, standard deviation	WFPf_cv_dur_std_phase	116
5	number of packets pause	WFPf_numP_pause	117
6	mean duration pause packets	WFPf_meadur_pause	118
7	duration of packets pause, standard deviation	WFPf_cv_dur_std_pause	119
8	ampli spec features	ampiSpecFeat1_1 - 28	81 - 108

Table 29: WFP features

2.3.6 RCS Features

For the RCS features the RCS values based on the raw signal and the lowpass signal are computed. The following three values are used as features:

- RCS value of the maximum of the lowpass signal
- RCS value of the maximum of the raw signal
- RCS value of the mean of the raw signal

#	Feature	DB-Name	ID
1	nearest RCS	RCS2_RCS_nearest_fea14	109
2	max lowpass RCS	RCS2_RCS_max_lowpassed	110
3	max raw RCS	RCS2_RCS_max_rawsignal	111
4	mean raw RCS	RCS2_RCS_mea_rawsignal	112

Table 30: RCS features

2.3.7 Low Frequency Features

The low frequency features are based on the spectrum computed via the STFT (mean over frequency bins) of the high-pass signal. The following values are used as features:

- Harmonic product spectrum (down sampling factors 1-4): frequency of the maximum peaks of the hps and its entropy
- Frequency bins 1-11 of the spectrum (low frequency bins)
- Frequency bins (width 2Hz) within a range (4-50Hz, medium frequency bins)

#	Feature	DB-Name	ID
1	Orientation/Phi	loHPSfundFreHz_1 - 4	159 - 162
2	mean RCS - A0	loHPSentropy_1 - 4	163 - 166
3	elongation -A2	spectrLowBins_1 - 11	122 - 132
4	cruciform element – A4	spectrMedBins_1 - 23	133 – 155

Table 31: low frequency features

2.3.8 Polarization / Shape Features

The polarization features are based on the raw and north signal and is only computed for echoes detected in modes with radar rotation.

The raw signal is split in 360° parts (indicated by the north signal). If the north signal is not available, the split can be approximated with the rotation frequency of the echo, given in the database. The mean over all splits is computed and normalized by its mean value. The parameters of the formula are fitted to the normalized mean split with a least square regression.

$$\|\sigma_{RCS}(\Omega t)\| = a_0 + a_2 \cdot \cos(2 \cdot (\Omega t - \varphi)) + a_4 \cdot \cos(4 \cdot (\Omega t - \delta))$$

$a_0: \sim \text{avg rcs} (= 1, \text{normalized})$
 $a_2: \sim \text{elongation}$
 $a_4: \sim \text{cruciform element}$
 $\varphi: \sim \text{orientation}$
 $\delta: \approx \varphi \text{ (insect symmetry axis)}$

Formula 1: polarization feature formula

The parameters of the best fit and its regression error are used as features.

#	Feature	DB-Name	ID
1	Orientation/Phi	shapeFeatPhi	169
2	mean RCS - A0	shapeFeatA0	170
3	elongation -A2	shapeFeatA2	171
4	cruciform element – A4	shapeFeatA4	172
5	regression error	shapeFeatRegrErr	173

Table 32: polarization/shape features

2.3.9 Wing Beat Frequency

The wing beat frequency is computed based on a subset of the features described in chapter 2.3 and saved itself as a feature. Along with the wing beat frequency a credibility value (0...1) is saved, quantifying the reliability of the wing beat frequency computation.

#	Feature	DB-Name	ID
1	wing beat frequency	WFF_predicted	167
2	credibility	WFF_credibility	168

Table 33: wing beat frequency and credibility

3 MR1Analysis Tool

3.1 Installation

1. Create a directory for the analysis.
2. Extract the MR1Analysis Scripts inside this directory
3. Open R-Studio, create a new Project inside this directory, name it with a meaningful name and save it.

The created directory should look like Figure 3.




	Code	13.04.2021 16:53	Dateiordner	
	Documentation	15.04.2021 15:49	Dateiordner	
	MR1Analysis.Rproj	15.04.2021 15:50	R Project	1 KB

Figure 3: new created project

3.2 Analyze Data

To analyze data, open the script AnalyzeData.R within the R-Project and set the parameters. After setting the parameters, the script can be sourced as one, or part of the code. The script is calling various functions to prepare and filter the data and computing the MTR. Some functions take a lot of time, depending on the amount of data. Therefore, on some point one might want to run only part of the script by filtering echoes or plotting data. The code is grouped with several `'if(TRUE) {...}'` respectively `'if(FALSE) {...}'` statements to easily exclude or include parts of the script while running it. At the beginning of the script the script Init.R has to be called to load all required libraries and initialize some global variables.

In the following chapters all the parts of the script AnalyzeData.R are described as well as the various functions.

3.2.1 Extract Data from Database

```

18 # ----- Load Data from database or file -----
19 # -----
20 # -----
21 dbServer <- "[your-SQL-server-name]"
22 dbName <- "[your-database-name]"
23
24 # Force Data to be extracted from DB
25 # if set to 'FALSE', data will anyway be extracted from the DB if data is not present yet.
26 forceToExtractDataFromDatabase = FALSE
27
28 # list of RF Feature ID's to extract. Feature ID's can be found in rfeatures table
29 # or in [data]$rFeatures once data was extracted from DB
30 # example to get wing beat frequency and credibility: c( 167, 168 )
31 listOfRfFeaturesToExtract <- c( 167, 168 )
32
33 # set timezones used by the radar and the target timezone
34 # Example: targetTimezone <- "Etc/GMT-1"
35 # --> Etc/GMT-1 equals UTC+1
36 # use "Etc/GMT0" (UTC) as radarTimezone for birdscan v1.6 and greater.
37 # birdscan v1.6 and greater stores all times as UTC.
38 radarTimezone <- "Etc/GMT0"
39 targetTimezone <- "Etc/GMT-1"
40
41 message( "Loading data" )
42 data <- extractDbData( dbServer = dbServer,
43                       dbName = dbName,
44                       radarTimezone = radarTimezone,
45                       targetTimezone = targetTimezone,
46                       forceToExtractDataFromDatabase = forceToExtractDataFromDatabase,
47                       listOfRfFeaturesToExtract = listOfRfFeaturesToExtract )
48

```

Figure 4: code snippet extract DB Data

As a first step the database data needs to be extracted. The function 'extractDbData' extracts the data from the database, arranges and merges the tables and saves the data to [your-project-directory]/Data/DB_Data. An Input Dialog will ask for the SQL Server credentials to log in to the SQL Server. Within the R-Project the extracted data will be available in the list called 'data'. The name of the list can of course be chosen freely, especially if data from multiple databases is extracted.

For date/time data the datatype 'POSIXct' is used which supports timezones. To add the correct timezones to the date/time data, the parameters 'radarTimezone' and 'targetTimezone' need to be set when calling the function 'extractDbData'. Use the "Etc" timezone format (e.g. UTC: "Etc/GMT0", UTC+1: "Etc/GMT-1, etc.)

- radarTimezone: The timezone of the date/time variables in the database set by the radar. For data created by BirdScan software v1.6 and higher, the timezone is always UTC (set to "Etc/GMT0")
- targetTimezone: The timezone for the analysis, the data will be converted to this timezone

Both timezone dates/times will be available in the data after extraction, indicated with the suffixes '_originTZ' and '_targetTZ'

Once the data is extracted and saved, recalling the function will not extract the data from the database, but load the saved file. Only if the function is forced to extract the data from the database by setting the parameter 'forceToExtractDataFromDatabase' to 'TRUE', the function will extract the data from the database and overwrite the already saved data.

Within the same project, data from different databases can be extracted by calling the function once for each database. The data will be saved in separate files named with the database name.

With the parameter 'listOfRfFeaturesToExtract' a vector with rf-feature IDs can optionally be passed to the function to specify which rfeatures should be extracted. See chapter 2.2.8 for more details and where to get the IDs. Note: Wing beat frequency and its credibility (IDs 167 and 168) are stored as rf-features.

3.2.2 Sunrise/Sunset

```

49 # .....
50 # ----- Sunrise/Sunset -----
51 # .....
52
53 # coordinates of site location (Latitude, Longitude )
54 # example: siteLocation <- c( 47.494427, 8.716432 )
55 siteLocation <- c( 47.494427, 8.716432 )
56
57 # set min and max time_stamps of echodata as timerange for sunrise/sunset calculation
58 timeRangeSunriseSunset <- c( min( data$echoData$time_stamp_targetTZ ),
59                             max( data$echoData$time_stamp_targetTZ ) )
60
61 message( "computing sunrise/sunset and twilight" )
62
63 # compute sunrise/sunset and dawn/dusk (civil)
64 sunriseSunset <- twilight( timeRange = timeRangeEchoData,
65                           latLon = siteLocation,
66                           timeZone = targetTimeZone )
67

```

Figure 5: code snippet sunrise/sunset

To group the data by day and night, the times of sunrise and sunset have to be computed for the time range. The function 'twilight' computes all sunrise and sunset times as well as the civil twilight times for a given time range and location.

Parameters for the function 'twilight':

- timeRange POSIXct vector of length 2, containing the start and end date/time of the time range for calculating sunrise and sunset times. To avoid complications, the time range has to be equal or exceed the time range of echo data.
- latLon a vector containing the latitude and longitude of the location of the radar, using decimal degrees format.
- timeZone the timezone of the sunrise/sunset times, usually the target timezone.

3.2.3 Add Day/Night Information to Echo Data

```

71 # .....
72 # ----- Add day/night information to echoData -----
73 # .....
74
75 sunOrCivil = "civil"
76
77 data$echoData <- addDayNightInfoPerEcho( echoData = data$echoData,
78                                         sunriseSunset = sunriseSunset,
79                                         sunOrCivil = sunOrCivil )
80 }
81

```

Figure 6: code snippet add day/night information

The function 'addDayNightInfoPerEcho' adds the two columns 'dayOrNight' and 'dateSunset' to the echo data. This information is not used any further by the script, but allows the user to filter echodata easily by "day" and "night".

Parameters for the function 'addDayNightInfoPerEcho':

- echoData dataframe with the echo data from the data list created by the function 'extractDBData'
- sunriseSunset dataframe with sunrise/sunset and civil twilight times created by the function 'twilight'
- sunOrCivil [optional] character string, "sun" (sunrise/sunset times) or "civil" (civil twilight times) to group by day and night. Default if not set is civil twilight.

3.2.4 Filter Protocol Data

```

84 # ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
85 # ----- Filter Protocols -----
86 # ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
87
88 # operation modes: pulselength (multiple selections possible)
89 # options: S, M, L
90 # example: pulseLengthSelection <- c( "S", "L" )
91 pulseLengthSelection <- c( "S" )
92
93 # operation modes: rotation (multiple selections possible)
94 # options: 1 (rotation), 0 (nonrotation)
95 # example: rotationSelection <- c( 1, 0 )
96 rotationSelection <- c( 1, 0 )
97
98 # subset protocolData by operation mode
99 message( "filtering protocol data")
100 protocolDataSubset <- filterProtocolData( protocolData = data$protocolData,
101                                           pulseTypeSelection = pulseLengthSelection,
102                                           rotationSelection = rotationSelection )
103

```

Figure 7: code snippet filter protocol data

With the function 'filterProtocolData' the protocol data can be filtered by the operation mode (pulse-type and antennarotation). The function returns the filtered subset of the protocolData which can later be used to filter the echoes based on the operation mode/protocol (see chapter 3.2.6).

- protocolData dataframe with the protocol data from the data list created by the function 'extractDBData'
- pulseTypeSelection [optional] character vector with the pulse types which should be included in the subset. Options: "S", "M", "L" (short-, medium-, long-pulse)
- rotationSelection [optional] numeric vector to select the operation modes with and/or without antennarotation. Options: 0, 1. (0 = no rotation, 1 = rotation)

3.2.5 Blind Times

```

104 # .....
105 # ----- Compute blindTimes -----
106 # .....
107
108 message( "loading and computing blindtimes")
109
110 # read manual blindtimes from csv file (filepath and -name: 'manualBlindTimesFile' defined in 'Init.R' )
111 # csv contains one row per blindtime and 3 columns (start of blindtime, stop of blindtime, type of blindtime)
112 # times have to be of format 'yyyy-MM-dd hh:mm:ss'
113 manualBlindTimes <- loadManualBlindTimes( filePath = manualBlindTimesFile,
114                                           blindTimesTZ = radarTimeZone,
115                                           targetTZ = targetTimeZone )
116
117 # compute blindtimes
118 blindTimes <- mergevisibilityAndManualBlindTimes( visibilityData = data$visibilityData,
119                                                    manualBlindTimes = manualBlindTimes,
120                                                    protocolData = protocolDataSubset )
121

```

Figure 8: code snippet load and compute blind times

Loading Manual Blind Times

For the MTR computation the times when the radar was blind have to be known. The radar itself can be blind in case of a protocol change (blocktime at the beginning of each protocol, usually 60s) or due to rain/snow or clutter (nearby objects, leaves or similar on radome, etc.). These times are stored in the visibility table or in the time_bins table in relation to the time bins duration (5min). To be flexible and not fixed to the 5min time bins created by the radar, the visibility table is used in this script.

In addition to the radar blind times, manual blind times can be defined. Manual blind times have to be defined in a csv file and loaded with the function 'loadManualBlindTimes'. A default file is available and can be edited: [your-project-directory]/Code/manualBlindTimes/manualBlindTimes.csv

The filepath is defined as a global variable 'manualBlindTimesFile'. A custom file and filepath can be used instead. The manual blind times have to be entered with 3 columns: start time [yyyy-mm-dd hh:mm:ss], stop time [yyyy-MM-dd hh:mm:ss], type.

Example:

2021-01-16 04:15:00,2021-01-16 05:42:00,rain

2021-01-17 16:33:00,2021-01-17 18:04:00,clutter

Manual blind time types can be chosen freely. When computing observation times (chapter 3.2.9), it can be decided if some of the defined manual blind time types should be treated as observed time with MTR zero or as blind time (e.g. rain).

If no file is present or the file is empty, no manual blind times will be computed.

Parameters for the function 'loadManualBlindTimes':

- filePath character string, absolute filepath of the manual blind time file.
- blindTimesTZ timezone of the blind times
- targetTZ target timezone of the blind times

Merging Radar and Manual Blind Times

For further processing the radar (visibility) and manual blind times have to be merged with the function 'mergeVisibilityAndManualBlindTimes'. This function will add a blind time type to the radar/visibility blind times. Blind times during the blocktime (usually 60s) at the beginning of each protocol are given the type "protocolChange", the rest of the radar blind times are given the type "visibility". After that the visibility and manual blind times will be merged. In case manual blind times and radar blind times are overlapping, radar blind times with type "visibility" will be overwritten, but not radar blind times with type "protocolChange".

Note: this function has to be called in any case, even if there are no manual blind times defined to create the dataframe with the blind times.

Parameters for the function 'mergeVisibilityAndManualBlindTimes':

- visibilityData dataframe with the visibility data from the data list created by the function 'extractDBData'.
- manualBlindTimes dataframe with the manual blind times created by the function 'loadManualBlindTimes'
- protocolData dataframe with the protocol data from the data list created by the function 'extractDBData' or a subset of it created by the function 'filterProtocolData'

3.2.6 Filter Echo Data

```

119 # ----- Filter Echoes -----
120 # -----
121 # -----
122
123 # classes used for analysis
124 # classes available listed in [data]$availableClasses
125 # example: classselection <- c("passerine_type", "wader_type", "large_bird")
126 classselection_allbirds <- c("passerine_type", "wader_type", "swift_type", "large_bird", "unid_bird", "bird_flock")
127 classselection_all <- c("passerine_type", "wader_type", "swift_type", "large_bird", "unid_bird", "bird_flock", "insect", "nonbio", "precipitation")
128
129 # classification probability cutoff (0..1)
130 classProbcutoff <- 0.3
131
132 # altitude range
133 altitudeRange_AGL_25_5000 <- c(25, 5000)
134
135 # time range for echodata (targetTimezone)
136 # use format "yyyy-MM-dd hh:mm"
137 timeRangeEchoData <- c("2020-01-01 00:00", "2021-01-01 00:00")
138 timeRangeEchoData <- as.POSIXct(timeRangeEchoData,
139                                format = "%Y-%m-%d %H:%M",
140                                tz = targetTimezone)
141
142 message("filtering echodata")
143
144 # subset echodata
145 echodataSubset_allbirds_25_5000 <- filterEchoData(echodata = data$echodata,
146                                                  timeRangeTargetTZ = timeRangeEchoData,
147                                                  protocolData = protocolDataSubset,
148                                                  classselection = classselection_allbirds,
149                                                  classProbcutoff = classProbcutoff,
150                                                  altitudeRange_AGL = altitudeRange_AGL_25_5000,
151                                                  manualBlindTimes = manualBlindTimes,
152                                                  echovalidator = TRUE)
153
154 echodataSubset_all_25_5000 <- filterEchoData(echodata = data$echodata,
155                                              timeRangeTargetTZ = timeRangeEchoData,
156                                              protocolData = protocolDataSubset,
157                                              classselection = classselection_all,
158                                              classProbcutoff = classProbcutoff,
159                                              altitudeRange_AGL = altitudeRange_AGL_25_5000,
160                                              manualBlindTimes = manualBlindTimes,
161                                              echovalidator = TRUE)

```

Figure 9: code snippet filter echo data

With the function 'filterEchoData' the echo data can be filtered by several parameters. The function returns the filtered echo data.

The Parameters of the function 'filterEchoData':

- **echoData** dataframe with the echo data from the data list created by the function 'extractDBData'.
- **timeRangeTargetTZ** POSIXct vector of length 2 with start and end time with target timezone. Echoes outside the time range will be excluded.
- **protocolData** dataframe with the protocol data from the data list created by the function 'extractDBData' or a subset of it created by the function 'filterProtocolData'. Echoes not detected during the listed protocols will be excluded.
- **classSelection** character string vector with the classes that should be included.
- **classProbCutOff** numeric cutoff value for class probabilities. Echoes with a lower class probability will be excluded.
- **altitudeRange_AGL** numeric vector of length 2 with start and end of the altitude range. Echoes outside the altitude range will be excluded.
- **manualBlindTimes** dataframe with the manual blind times created by the function 'loadManualBlindTimes'.
- **echoValidator** logical, if set to TRUE, echoes labelled by the echo validator as "non-bio scatterer" will be excluded. If set to FALSE, all echoes are included.

3.2.7 Altitude Bins

```

161 # .....
162 # ----- Create Altitude Bins -----
163 # .....
164
165 # Altitude Range and Bin Size
166 altitudeRange_AGL_25_1025 <- c( 25, 1025 )
167
168 # create altitudeBins
169 message( "creating altitude bins" )
170 altitudeBins_25_1025_oneBin <- createAltitudeBins( altitudeRange = altitudeRange_AGL_25_1025,
171                                                    altitudeBinSize = 1000 )
172 altitudeBins_25_1025_binSize50 <- createAltitudeBins( altitudeRange = altitudeRange_AGL_25_1025,
173                                                         altitudeBinSize = 50 )
174

```

Figure 10: code snippet create altitude bins

The MTR values will be computed per time bin and altitude bin. The function 'createAltitudeBin' prepares a dataframe containing the altitude bins with a given size and within an altitude range.

Parameters for the function 'createAltitudeBin':

- **altitudeRange** numeric vector of length 2 with the start and end of the altitude range in meter a.g.l.
- **altitudeBinSize** numeric, size of the altitude bins in meter.

3.2.8 Time Bins

```

179 # ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
180 # ----- Create Time Bins -----
181 # ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
182
183 # time range for timebins (targetTimeZone)
184 # use format "yyyy-MM-dd hh:mm"
185 timeRangeTimeBins <- c( "2020-03-01 00:00", "2020-11-01 00:00" )
186 timeRangeTimeBins <- as.POSIXct( timeRangeTimeBins,
187                                 format = "%Y-%m-%d %H:%M",
188                                 tz = targetTimeZone )
189
190 # timebin size in seconds
191 timeBinDuration_sec <- 3600
192
193 # create Timebins
194 message( "creating time bins" )
195 timeBins_1h <- createTimeBins( timeRange = timeRangeTimeBins,
196                               timeBinDuration_sec = timeBinDuration_sec,
197                               timeZone = targetTimeZone,
198                               sunriseSunset = sunriseSunset,
199                               sunOrCivil = sunOrCivil )
200

```

Figure 11: code snippet create time bins

The MTR values will be computed per time bin and altitude bin. The function 'createTimeBins' prepares a dataframe containing the time bins with a given duration and within a time range. Time bins starting before and ending after a sunrise or sunset (or civil twilight) will be split in two bins, at the time of sunrise/sunset.

Parameters for the function 'createTimeBins':

- timeRange POSIXct vector of length 2 with start and end time with target timezone. Echoes outside the time range will be excluded.
- timeBinDuration_sec numeric, time bin duration in seconds
- timeZone character string, timezone of the time bins -> target timezone
- sunriseSunset dataframe with the sunrise/sunset times and the civil twilight times created by the function 'twilight'.
- sunOrCivil [optional] character string, "sun" (sunrise/sunset times) or "civil" (civil twilight times) to group by day and night. Default if not set is civil twilight.

3.2.9 Observation Times

```

197 # ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
198 # ----- Compute Observation Time for each Time Bin -----
199 # ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
200
201 # set blindtime types which should not be treated as blindtime but MTR = 0
202 blindTimeASmtrZero <- c( "rain" )
203
204 # compute observation times
205 message( "computing observation times" )
206 timeBins_1h <- computeObservationTime( timeBins = timeBins_1h,
207                                       protocolData = protocolDataSubset,
208                                       blindTimes = blindTimes,
209                                       blindTimeASmtrZero = blindTimeASmtrZero )
210

```

Figure 12: code snippet compute observation times

To compute MTR values normalized to one hour (individuals/km/hour), the observation time within each time bin has to be known. To calculate the observation time, the operation time of the radar during each time bin has to be known, as well as the blind time during the operation time.

The function `'computeObservationTime'` computes this observation time for each time bin.

With the parameter `'blindTimeAsMtrZero'` it is possible to not treat certain blind time types as blind time, but as observation time with MTR zero.

For example, during periods of rain the radar is usually blind by itself (indicated in visibility table) but not always. During light rain the radar might not switch to blind but detects a lot of clutter.

Therefore, it can make sense to register periods of rain manually as manual blind time with type "rain". It is now possible to treat periods of rain as blind time which results in not available MTR values during these periods. If the periods of rain are treated as observation time with MTR zero (assuming no birds/insects are flying during rain), it will result in MTR values of zero. Both ways are justifiable.

Parameters of the function `'computeObservationTime'`:

- `timeBins` dataframe with the time bins created by the function `'createTimeBins'`.
- `protocolData` dataframe with the protocol data from the data list created by the function `'extractDBData'` or a subset of it created by the function `'filterProtocolData'`.
- `blindTimes` dataframe containing the blind times created by the function `'mergeVisibilityAndManualBlindTimes'`.
- `blindTimesAsMtrZero` character string vector with the blind time types which should be treated as observation time with MTR zero.

3.2.10 MTR Computation

```

211 # ----- Compute MTR -----
212 # ----- Compute MTR -----
213 # ----- Compute MTR -----
214
215 # cutoff for proportional observation times: ignore timebins where
216 # "observationTime/timeBinDuration < propObsTimeCutoff"
217 # in 'computeMTR' only used if parameter 'computePerDayNight' is set to
218 # 'TRUE' and timebins are shorter than day/night. In this case timebins
219 # are combined by day/night and only time bins with a proportional
220 # observation time greater than the cutoff will be used to compute the
221 # day/night MTR and spread.
222 # set value from 0-1
223 propObsTimeCutoff <- 0.2
224
225 message( "computing MTR per day/night, 25m to 1025m" )
226 mtr_DayNight_25mto1025m <- computeMTR( echoes = echoDataSubset_allBirds_25_5000,
227                                       classSelection = classSelection_allBirds,
228                                       altitudeBins = altitudeBins_25_1025_oneBin,
229                                       timeBins = timeBins_1h,
230                                       propObsTimeCutoff = propObsTimeCutoff,
231                                       computePerDayNight = TRUE )
232
233 saveMTR( mtr = mtr_DayNight_25mto1025m,
234          filepath = mtrDataDir,
235          dbName = dbName,
236          rotSelection = rotationSelection,
237          pulseTypeSelection = pulseLengthSelection )
238
239 # MTR per hour, one altitude bin 25m to 1000m
240 message( "computing MTR per hour, 25m to 1025m" )
241 mtr_1h_25mto1025m <- computeMTR( echoes = echoDataSubset_allBirds_25_5000,
242                                 classSelection = classSelection_allBirds,
243                                 altitudeBins = altitudeBins_25_1025_oneBin,
244                                 timeBins = timeBins_1h,
245                                 propObsTimeCutoff = propObsTimeCutoff,
246                                 computePerDayNight = FALSE )
247
248 saveMTR( mtr = mtr_1h_25mto1025m,
249          filepath = mtrDataDir,
250          dbName = dbName,
251          rotSelection = rotationSelection,
252          pulseTypeSelection = pulseLengthSelection )
253
254 # MTR per day/night, altitude bins of size 50m from 25m to 1025m
255 message( "computing MTR per day/night, 25m to 1025m, 50m altitude bins" )
256 mtr_DayNight_25mto1025m_50m <- computeMTR( echoes = echoDataSubset_allBirds_25_5000,
257                                             classSelection = classSelection_allBirds,
258                                             altitudeBins = altitudeBins_25_1025_binSize50,
259                                             timeBins = timeBins_1h,
260                                             propObsTimeCutoff = propObsTimeCutoff,
261                                             computePerDayNight = TRUE )
262
263 saveMTR( mtr = mtr_DayNight_25mto1025m_50m,
264          filepath = mtrDataDir,
265          dbName = dbName,
266          rotSelection = rotationSelection,
267          pulseTypeSelection = pulseLengthSelection )
268
269 # MTR per hour, altitude bins of size 50m from 25m to 1025m
270 message( "computing MTR per hour, 25m to 1025m, 50m altitude bins" )
271 mtr_1h_25mto1025m_50m <- computeMTR( echoes = echoDataSubset_allBirds_25_5000,
272                                     classSelection = classSelection_allBirds,
273                                     altitudeBins = altitudeBins_25_1025_binSize50,
274                                     timeBins = timeBins_1h,
275                                     propObsTimeCutoff = propObsTimeCutoff,
276                                     computePerDayNight = FALSE )
277
278 saveMTR( mtr = mtr_1h_25mto1025m_50m,
279          filepath = mtrDataDir,
280          dbName = dbName,
281          rotSelection = rotationSelection,
282          pulseTypeSelection = pulseLengthSelection )

```

Figure 13: code snippet compute MTR

Computing MTR

Computing MTR is time consuming. There are two options to compute MTR values. The options are selected with the parameter 'computePerDayNight'.

Option	Description
--------	-------------

compute MTR for each time bin	This option computes the MTR for each time bin defined in the time bin dataframe. The time bins that were split due to sunrise/sunset during the time bin will be combined to one bin.
compute MTR per day/night	The time bins of each day and night will be combined and the mean MTR is computed for each day and night. Aside this, the spread (first and third Quartile) for each day and night is computed. The spread is dependent on the chosen time bin duration/amount of time bins.

In the above code snippet (Figure 13), four different MTR tables are created:

- MTR per day and night and one altitude bin between 25m and 1025m.
- MTR per time bin (1h) and one altitude bin between 25m and 1025m.
- MTR per day and night and multiple altitude bins of size 50m between 25m and 1025m.
- MTR per time bin (1h) and multiple altitude bins of size 50m between 25m and 1025m.

Parameters of the function 'computeMTR':

- **echoes** dataframe with the echo data from the data list created by the function 'extractDBData' or a subset of it created by the function 'filterEchoData'.
- **classSelection** character string vector with all classes which should be used to calculate the MTR. The MTR and number of Echoes will be calculated for each class as well as for all classes together.
- **altitudeBins** dataframe with the altitude bins created by the function 'createAltitudeBins'. MTR is computed for each altitude bin.
- **timeBins** dataframe with the time bins created by the function 'computeObservationTime'. MTR is computed for each time bin.
- **propObsTimeCutoff** numeric between 0 and 1. If the MTR is computed per day and night, time bins with a proportional observation time smaller than propObsTimeCutoff are ignored when combining the time bins. If the MTR is computed for each time bin, the parameter is ignored.
- **computePerDayNight** logical, TRUE: MTR is computed per day and night
FALSE: MTR is computed for each time bin

Saving MTR

Since computing MTR can take a lot of time (depending on the amount of time bins, altitude bins and number of echoes), the tables with the MTR values can be saved as rds-files. To save MTR tables the function 'saveMTR' can be used. The function creates a filename that specifies:

- database name
- timerange date from/to (yyyymmdd)
- time bin size (in seconds or "day/night")
- altitude range from/to
- number of altitude bins
- abbreviations of classes (classSelection)

- rotation selection
- pulse type selection

In the above code snippet (Figure 13), the files are saved to the directory [your-project-directory]/Data/MTR, the path is available as a global variable 'mtrDataDir'. A custom path can be set alternatively.

Parameters of the function 'saveMTR':

- mtr dataframe with MTR values created by the function 'computeMTR'
- filepath character string, path of the directory, e.g. "[your-project-directory]/Data/MTR"
- dbName character string, name of the database
- rotSelection numeric vector, rotation selection which was used to filter protocols. Used to create the filename.
- pulseTypeSelection character vector, pulse type selection which was used to filter protocols. Used to create the filename.

The saved MTR data can be read with the R-function 'readRDS()':

```
mtr_data <- readRDS( filepath )
```

3.2.11 Plot MTR

The created data (MTR, echoes) can be plotted in various ways and graphs. Two examples are implemented within this script.

```

285 # ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
286 # ----- Plot -----
287 # ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
288
289 # time range for plots (targetTimeZone)
290 # use format "yyyy-MM-dd hh:mm"
291 timeRangePlot <- list( as.POSIXct( c( "2020-03-01 00:00", "2020-04-01 00:00" ),
292                               format = "%Y-%m-%d %H:%M", tz = targetTimeZone ),
293                       as.POSIXct( c( "2020-04-01 00:00", "2020-05-01 00:00" ),
294                               format = "%Y-%m-%d %H:%M", tz = targetTimeZone ),
295                       as.POSIXct( c( "2020-05-01 00:00", "2020-06-01 00:00" ),
296                               format = "%Y-%m-%d %H:%M", tz = targetTimeZone ),
297                       as.POSIXct( c( "2020-03-01 00:00", "2020-11-01 00:00" ),
298                               format = "%Y-%m-%d %H:%M", tz = targetTimeZone ))
299
300 # plot longitudinal MTR per day and night
301 if( FALSE )
302 {
303   message( "plot longitudinal MTR (all classes in MTR data)" )
304   plotLongitudinalMTR( mtr = mtr_DayNight_25mto1025m,
305                       maxMTR = -1,
306                       timeRange = timeRangePlot,
307                       plotClass = "allclasses",
308                       propobsTimeCutoff = propobsTimeCutoff,
309                       plotspread = TRUE,
310                       filePath = plotDir )
311   message( "plot longitudinal MTR (passerine_type)" )
312   plotLongitudinalMTR( mtr = mtr_DayNight_25mto1025m,
313                       maxMTR = -1,
314                       timeRange = timeRangePlot,
315                       plotClass = "passerine_type",
316                       propobsTimeCutoff = propobsTimeCutoff,
317                       plotspread = TRUE,
318                       filePath = plotDir )
319 }
320
321 # Exploration plot with all classes up to 5000m.
322 if( FALSE )
323 {
324   message( "plot exploration" )
325   plotExploration( echoData = echoDataSubset_all_25_5000,
326                   timeRange = timeRangePlot,
327                   manualBlindtimes = manualBlindtimes,
328                   visibilityData = data$visibilityData,
329                   protocolData = protocolDataSubset,
330                   sunriseSunset = sunriseSunset,
331                   maxAltitude = -1,
332                   filePath = plotDir )
333 }
334

```

Figure 14: code snippet Plots

MTR per Day and Night

The function 'plotLongitudinalMTR' plots the daily MTR values per day and night as a bar plot (Figure 15). For each bar the spread (first and third Quartile) is shown as error bars as well as the numbers of echoes. Periods with no observation are indicated with grey, negative bars.

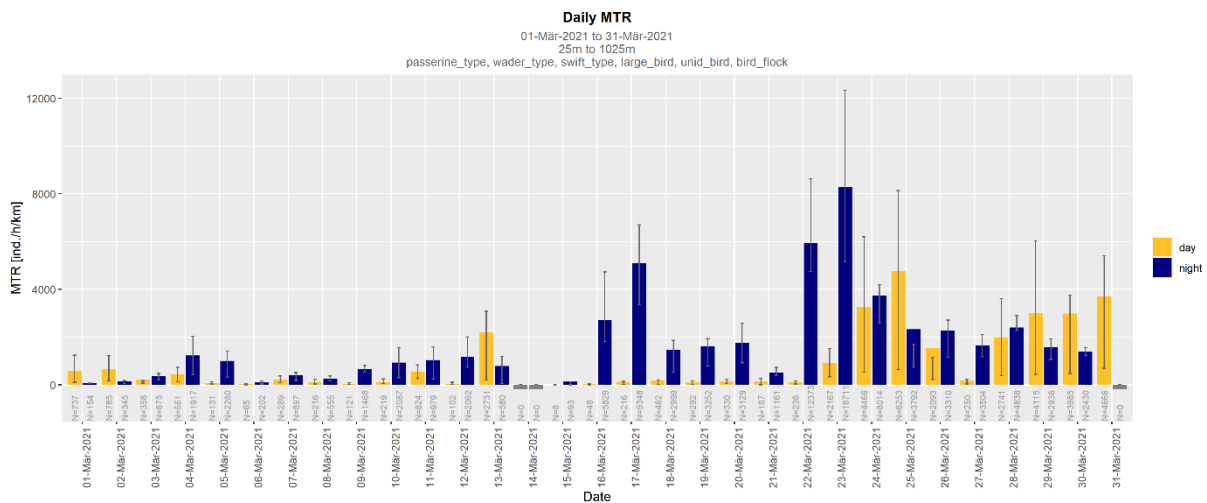


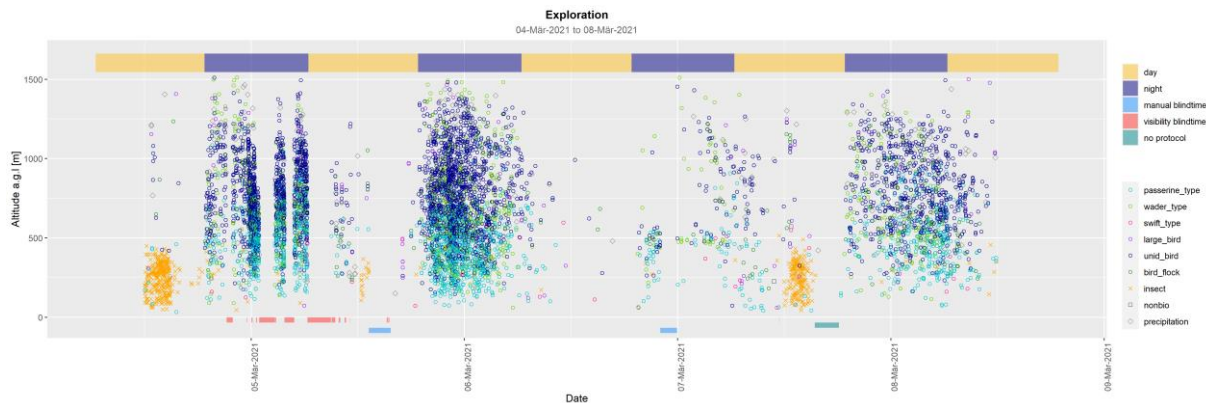
Figure 15: daily MTR plot

Parameters for the function `plotLongitudinalMTR`:

- `mtr` dataframe with MTR values created by the function `computeMTR`.
The MTR data should be computed with one altitude bin, if MTR data with multiple altitude bins are passed to the function, the only the lowest altitude bin is plotted.
The MTR data should be computed per day and night (setting the parameter `computePerDayNight` of the function `computeMTR` to TRUE, otherwise the plot will fail.
- `maxMTR` [optional] numeric, fixes the maximum value of the y-Scale of the plot to the given value. If negative or not set, the y-Scale is auto-scaled.
- `timeRange` [optional] list of POSIXct vectors length 2, start and end time of the timeranges that should be plotted. The date/time format is "yyyy-MM-dd hh:mm". Target timezone has to be added to the times (Figure 14 as example). If not set, all MTR data is plotted in one plot.
- `plotClass` character string with the class of which the MTR data should be plotted. If not set or set to "allClasses", MTR of all classes will be plotted.
- `plotSpread` logical, choose if the spread (first and third quartile) should be plotted
- `filePath` character string, path of the directory where the plot should be saved. The function `savePlotToFile` is used to save the plots as pngs with an autogenerated filename.

Plot Echo Data

The function 'plotExploration' creates a scatter plot with the altitude of the echoes over time. The different classes are indicated with different colours and shapes. Additionally, day/night periods as well as blind times (manual and radar/visibility) and not observed times (no protocol due to filtered operation mode or radar was not operating) are shown.



Parameters for the function 'plotExploration':

- **echoData** dataframe with the echo data from the data list created by the function 'extractDBData' or a subset of it created by the function 'filterEchoData'.
- **timeRange** [optional] list of POSIXct vectors length 2, start and end time of the timeranges that should be plotted. The date/time format is "yyyy-MM-dd hh:mm". Target timezone has to be added to the times (Figure 14 as example). If not set, all echo data is plotted in one plot.

Note: Too long time-ranges may produce an error if the created image is too large and the function can't allocate the file.
- **manualBlindTimes** [optional] dataframe with the manual blind times created by the function 'loadManualBlindTimes'. If not set, manual blind times are not shown in the plot.
- **visibilityData** [optional] dataframe with the visibility data created by the function 'extractDBData'. If not set, visibility data are not shown in the plot.
- **protocolData** [optional] dataframe with the protocol data used to filter the echoes, created by the function 'extractDBData' or a subset of it created by the function 'filterProtocolData'. If not set, periods without a protocol are not shown in the plot.
- **sunriseSunset** [optional] dataframe with sunrise/sunset and civil twilight times created by the function 'twilight'. If not set, day/night times are not shown in the plot.
- **maxAltitude** [optional] numeric, fixes the maximum value of the y-Scale of the plot to the given value. If negative or not set, the y-Scale is auto-scaled.

-
- `filePath` character string, path of the directory where the plot should be saved. The function `'savePlotToFile'` is used to save the plots as pngs with an autogenerated filename.