Development and implementation of a QGIS-Plugin for download, processing and visualization of MODIS LST satellite data

Bachelor thesis

To obtain the academic degree

Bachelor of Science

Martin-Luther-Universität Halle-Wittenberg

Faculty of Science III

Institute of Earth Sciences and Geography

Department of Digital Geography

First reviewer: M. Sc. Max Hörold, Digital Geography

Second reviewer: M. Sc. Hannes Hemmerle, Applied Geology

Submitted by

Jonas Apelt

Matriculation number: 217203070

Halle (Saale), 25-Apr-21

Statement of authorship

I hereby certify that I have written this thesis independently and have not used any sources other than those listed in the bibliography. All passages that are taken verbatim from published sources are marked as such. The drawings or illustrations in this thesis have been made by myself or are provided with a corresponding source reference.

Jonas Apelt

____

Halle (Saale), Sunday, April 25, 2021

Table of content

List of abbreviations

API – Application User Interface

CWV – Column Water Vapor

ESDT – Earth Science Data Type

HDF – Hierarchical Data Format

IFOV – Instantaneous Field of View

LANDSAT – Land Remote-Sensing Satellite

LSE – Land surface emissivity

LST – Land Surface Temperature

MIR – mid-infrared

MODIS – Moderate-resolution Imaging Spectroradiometer

MODTRAN3 – Moderate resolution atmospheric Transmission

NASA – National Aeronautics and Space Administration

NDVI – Normalized Difference Vegetation Index

NEDT – noise equivalent differential temperature

SDS – Scientific Datasets

SST – Sea Surface Temperature

TIFF – Tagged Image File Format

TIR – Thermal infrared

UHI – Urban Heat Islands

UI – User Interface

USGS – United States Geological Survey

WKT – Well-Known-Text Format

List of symbols

$B_\lambda(T)$ – Spectral emission density of the surface

K_1 – Calibration constant one

K_2 – Calibration constant two

$L_\lambda(T_s)$ – Radiance of a black body at temperature T_s

$L_{\lambda down}$ – Down-welling radiance

$L_{\lambda up}$ – Up-welling radiance

T_4, T_5 – Radiance values of bands 31 and 32

T_{air} – Air Temperature

T_s – Surface Temperature

°*C* – Degree Celsius

°*F* – Degree Fahrenheit

°*K* – Degree Kelvin

A, B, C – Algorithm coefficients

c – Speed of light

cm – Centimeter

h – Planck constant

J – Joule

k – Boltzmann constant

L – Spectral radiance

sr – Steradian

T – Temperature

ε – Land surface emissivity

$\varepsilon(\lambda)$ – Spectral emissivity

λ – Wavelength

μm – Micrometer

τ – Atmospheric transmittance

1. Introduction

Weather observations, and the weather forecasts derived from them, provide information so indispensable for facilitating everyday life as to be taken for granted: particularly ubiquitous are the measurements of air and surface temperatures and the influence that this collective information exerts. In the context of ever-increasing worldwide urbanization and the targeted improvement of public and urban quality of life, this data can provide a starting point when planning urban spaces. The phenomenon of urban heat islands in particular has become increasingly topical in current scientific discourse, as urban planners can benefit from understanding how surface temperatures can be influenced by certain building forms. In order to provide sufficient data for this, large-area raster data with the highest possible resolution is needed, as conventional weather measuring stations not only require and impractically vast (and therefore costly) measuring network, but also do not provide reliable results even after interpolation. As such, remote sensing methods with thermal infrared sensors (TIR) offer a far more feasible possibility of covering large areas with satellites.

Therefore, this work was dedicated to the development of a QGIS plugin that can download, process and visualize Moderate-resolution Imaging Spectroradiometer (MODIS) Land Surface Temperature (LST) data, thus avoiding manual downloading and processing via the Earth Explorer of the United States Geological Survey (USGS) ("EarthExplorer," 2021): this is a great benefit for the user both because it can be tricky to access the relevant data of interest from the abundance of offer and because the existing Hierarchical Data Format (HDF) is not directly readable in QGIS or ArcGIS.

The practical part of this work should enable the user to select a specific section for any time period directly in QGIS via the selected map section or via the designation of the tiles. Furthermore, it should be possible to choose between two product types: on the one hand the MOD11A1 with a pixel size of 1km and a temporal resolution of one day, and on the other hand the MOD11A2 product, which also provides a 1km pixel size every eight ("MODIS Land Team Home Page," 2021). In addition, the user should be able to make a selection of the desired MODIS satellites MODIS Terra and MODIS Aqua, as well as the option of deleting all superfluous data. To distinguish downloads, a suffix can be added to the file name. If the user has selected the

MOD11A1 product, all data (i.e. that of the night results as well as that of the days over any given period) are combined to an average value in a new file. To validate the data, another file is created for each of these mean values, which expresses in percent how many days of the total period were missing values.
In the theoretical part of the paper, only the basic information about the satellite, the algorithm for processing the sensor data and, finally the software tool itself will be discussed.

2. MODIS LST

Wan et al. (1997) point out clearly that MODIS provides an excellent opportunity to develop a physics-based algorithm to simultaneously retrieve surface emissivity and temperature, due to the multiple bands in the mid-infrared (MIR) range and in the 8–14 m window. MODIS is particularly useful because of its global coverage, radiometric resolution and dynamic ranges, as well as its accurate calibration in multiple TIR bands designed for retrievals of Sea Surface Temperature (SST), LST and atmospheric properties. Specifically, bands 3-7, 13, and 16-19 are used to classify land-cover from which emissivity is inferred; band 26 for detecting cirrus clouds, and TIR bands 20, 22, 23, 29, 31, and 32 correct for atmospheric effects and retrieve Land surface emissivity (LSE) and temperature. For calibration, MODIS views cold space and an artificial blackbody-like calibration tool before viewing the earth scene thus achieving an accuracy of more than 99%. (Zhengming Wan and Dozier, 1996)
MODIS is a sensor that is installed in the satellites Terra (EOS AM-1) and Aqua (EOS PM-1), among others. Terra and Aqua orbit the Earth's poles in a sun-synchronous manner at a constant temporal interval at an altitude of 705km, so that Terra moves from north to south in the morning and Aqua in the afternoon. Terra reaches the North Pole at 10:30 before beginning its descent south, whereas Aqua begins its ascent north at 13:30. Both satellites cover the entire Earth with a swath width of 2330km every one to two days. The total of 36 bands offer a spatial resolution of 250m (bands 1-2), 500m (bands 3-7) and 1000m (bands 8-36) in a spectral resolution from 620nm (band 1) to 14385nm (band 36). Bands 31 and 32, whose wavelengths measure 10,780 - 11,280 µm and 11,770 - 12,270 µm respectively, are used to estimate the surface temperatures

of the Earth. A detailed band specification is given in Table 1. For further information please visit NASAS website about MODIS ("MODIS Web," 2020).

band	bandwidth (nm)	IFOV	primary use	band	bandwidth (nm)	IFOV	primary use	NEDT (°K)
1	620-670	250m	L	19	915-965	1km	A	
2	841-876	250m	A,L	**20**	**3660-3840**	**1km**	**O, L**	**0.05**
3	459-479	500m	L	21	3929-3989	1km	fire, vol-cano	
4	545-565	500m	L	**22**	**3929-3989**	**1km**	**A, L**	**0.07**
5	1230-1250	500m	L	**23**	**4020-4080**	**1km**	**A, L**	**0.07**
6	1628-1652	500m	A,L	24	4433-4498	1km	A	0.25
7	2105-2155	500m	A,L	25	4482-4549	1km	A	0.25
8	405-420	1km	O	26	1360-1390	1km	cirrus	
9	438-448	1km	O	27	6535-6895	1km	A	0.25
10	483-493	1km	O	28	7175-7475	1km	A	0.25
11	526-536	1km	O	**29**	**8400-8700**	**1km**	**L**	**0.05**
12	546-556	1km	O	30	9580-9880	1km	ozone	0.25
13	662-672	1km	O	**31**	**10780-11280**	**1km**	**A, L**	**0.05**
14	673-683	1km	O	**32**	**11770-12270**	**1km**	**A, L**	**0.05**
15	743-753	1km	O	**33**	**13185-13485**	**1km**	**A, L**	**0.25**
16	862-877	1km	O	34	13485-13785	1km	A	0.25
17	890-920	1km	A	35	13785-14085	1km	A	0.25
18	931-941	1km	A	36	14085-14385	1km	A	0.35

Table 1: Band specifications for MODIS (Wan, 2013)

A - Atmospheric studies; L - Land studies; O - Ocean studies.

2.1 Physical basics

The basic physical assumptions about LSE from TIR are briefly presented here. Emitted spectral radiance L at wavelength λ from a surface at thermodynamic temperature T_s, is given by multiplying the Planck function by spectral emissivity $\varepsilon(\lambda)$. (Zhengming Wan and Dozier, 1996)

$$L(\lambda.T) = \varepsilon(\lambda)B(\lambda,T_s) \qquad (1)$$

2.1.1 Plancks law

Planck's law describes the spectral emission density of a black body in dependence of wavelength λ and the absolute temperature T and can be expressed as the following (Humes et al., 1994):

$$B_\lambda(T) = \frac{2\pi hc^2}{\lambda^5\ (e^{\frac{hc}{\lambda kT}} - 1} \tag{2}$$

With $B_\lambda(T)$ as spectral emission density of the surface in dependence of the temperature ($Wm^{-2}\ \mu m^{-1}\ sr^{-1}$), λ as wavelength in m, h as the Planck constant (h = 6.626076 x 10-34 Js), k as the Boltzmann constant (k = 1.380658 10^{-23} JK^{-1}), T as the temperature in K and c as the speed of light (c = 2.99792458 x 10^8 ms^{-1}).

2.1.2 Wien's law of displacement

Figure 1 shows the proportional dependence of emitted radiance and temperature of a black body according to Planck's law: As the temperature increases, the maximum radiation density tends towards smaller wavelengths. This phenomenon is also known as Wien's law of displacement. Natural sunlight with an average temperature of 5778K has its peak at about 0.55 μm in the range of visible light, while the earth's emission is in the spectrum of TIR. Therefore within an atmospheric window of 8-14 µm, Land surface temperature estimations with TIR data become possible. (Heipke et al., 2017)

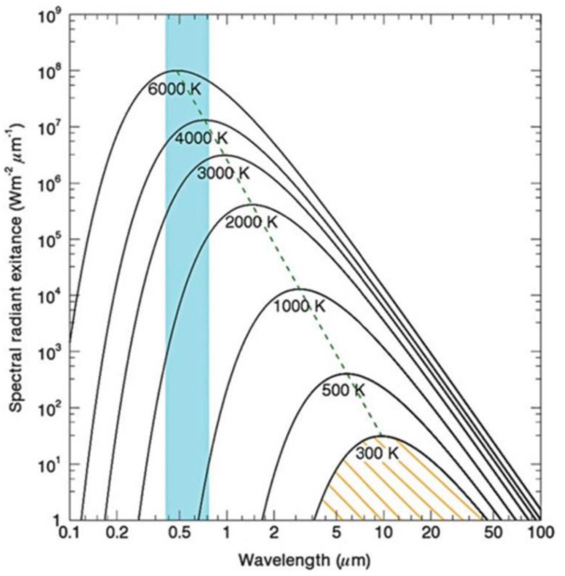


Figure 1: Blackbody radiation curves at different selected temperatures (Kuenzer und Dech 2013)

Figure 2 shows the transmission properties of the atmosphere as a whole, as well as the transmission functions of selected gases such as N_2, CO_2 and H_2O calculated with MODTRAN3 (Moderate resolution atmospheric Transmission) (Berk, 1987). The selected scenario was recorded under typical conditions at a viewing angle of 45°, a visibility of 23km and midsummer conditions of 2.9 cm column water vapour (CWV). Scattering by aerosols and molecules was also taken into account here. Due to its stable transmissivity around 0.95-0.98 an average distribution in the atmospheric radiative model is sufficient. Figure 2 is intended to give an impression of where MODIS bands are located in the electromagnetic spectrum along atmospheric windows and transmissivities.

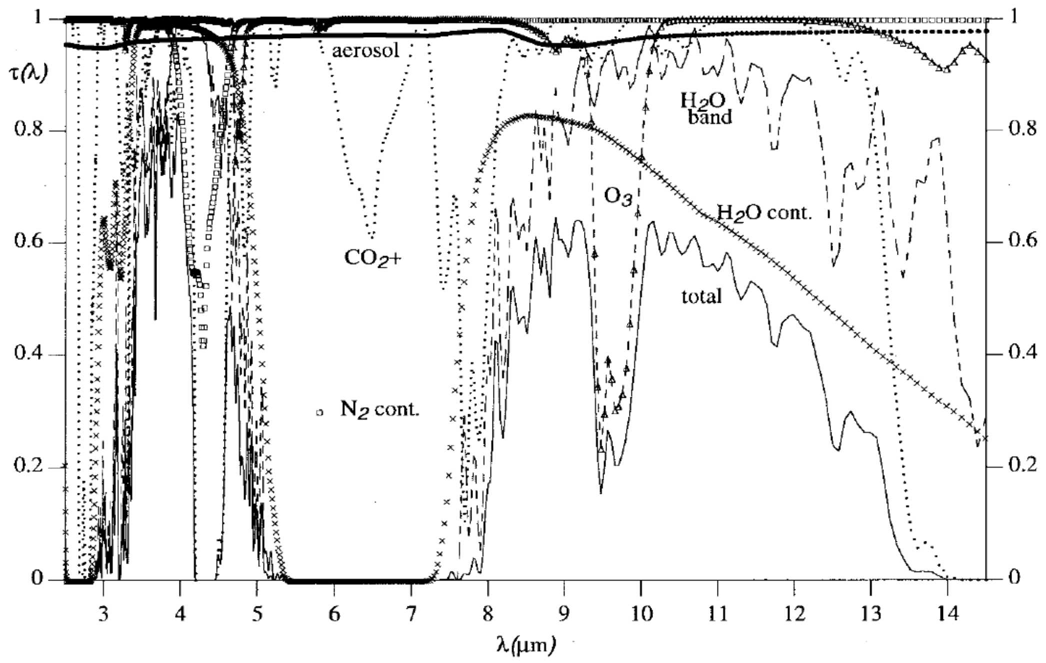


Figure 2: Atmospheric transmission functions at view angle 45° and in mid-latitude summer condition (column water vapor 2.9 cm, visibility 23 km); calculated with MODTRAN3 (Zhengming Wan and Dozier, 1996)

2.1.3 Radiative Transfer Model

The radiative transfer method can be expressed according to the following principle (Isaya Ndossi and Avdan, 2016):

$$L_\lambda = [\varepsilon_\lambda L_\lambda(T_s) + (1 - \varepsilon_\lambda) L_{\lambda down}]\tau + L_{\lambda up} \tag{3}$$

L_λ is the TIR radiance received by the satellite, which is mainly influenced by surface radiation, up-welling radiance into the atmosphere ($L_{\lambda up}$) and down-welling radiance ($L_{\lambda down}$) from the atmosphere. τ is the atmospheric transmittance, ε is LSE and $L_\lambda(T_s)$ is the radiation of a black body at temperature T_s. After determining the atmospheric conditions, for example with a National Aeronautics and Space Administration (NASA) calculation tool or other MODIS data, the surface temperature can be calculated. (Isaya Ndossi and Avdan, 2016)

$$L_\lambda\,(T_s) = \frac{L_\lambda - L_{\lambda up}}{\tau \varepsilon \lambda} - \frac{1 - \varepsilon_\lambda}{\varepsilon_\lambda}\, L_{\lambda down} \tag{4}$$

After the radiance has been determined, the result is converted to LST with similar relationships and two calibration constants K_1 and K_2 similar to the Planck function (Isaya Ndossi and Avdan, 2016).

$$T_s = \frac{K_2}{\ln\left(\frac{K_1}{L_\lambda(T_s)} + 1\right)} \tag{5}$$

2.2 LST algorithm

To estimate LST from space three types of methods have been developed: the single-infrared channel method, the split-window method (which is used in various multi-channel SST algorithms), and a day/night MODIS LST method which is designed to take advantage of the unique design of the MODIS instrument adopted to scientific usage on land and on sea. The single-channel method relies on a good radiative transfer model and comprehensive atmospheric profiles, which must be given by either satellite soundings or conventional radiosonde. (Jiménez-Muñoz and Sobrino, 2003)

The second method, which doesn't require exact previous knowledge about surface emissivity, was applied to the MOD11B product and its variations. It uses day/ night pairs of TIR data to calculate LST and band-averaged emissivity at the same time without knowing atmospheric temperature and water vapor profiles precisely. Band 26 detects cirrus clouds, just as all atmospheric channels listed in table 1 are used to retrieve atmospheric temperature and water vapor profiles. Like the split window algorithm, the day/night algorithm corrects for atmospheric effects to gain information about surface temperature and emissivity from TIR bands 20, 22, 23 (3.5-4.2 μm atmospheric window) and 29, 31-33 (8-14 μm atmospheric window). Moreover, precision for day time LST is increased by MIR bands to better differentiate between surface emission and solar radiation. Aerosols and molecules are only considered in the MODTRAN model due to their relative constant transmissivity between 0.95-0.98. (Zhengming Wan and Zhao-Liang Li, 1997)

The split-window algorithm uses two adjacent infrared bands (Bands 31 and 32 in MODIS) to correct atmospheric effects based on the differential absorption. After the development of various SST algorithms, LST algorithm retrieval garnered greater scientific attention. While sea surfaces have very low local differences in emissivity, land covers are more diverse. In simulations it is possible to gain knowledge about certain surfaces such as lakes, snow and ice, bare soil or rainforest canopies. However, arid and semi-arid terrains represent a significant challenge when using split-window LST methods given the high variability of land covers within short periods of time. (Zhengming Wan and Zhao-Liang Li, 1997)

Over the course of the aforementioned algorithm's development, several major improvements have been made, such as the inclusion of CWV over temperature ranges for coefficient optimization (Sobrino et al., 1993) as well as implementing view angles in the LST algorithm (Wan, 1999). In addition, for view angles above 45° it is necessary to subdivide the surface temperatures, the CWV and the lower boundary temperature into further subranges. Second of all, Land surface emissivity vary widely on Land surface properties. As a result it becomes necessary to analyze emissivity in a wide range of samples to improve the algorithm's output. As Salisbury and D'Aria (1992) analyze LSE in their paper for a wide variety of materials (79 samples), LSE is principally retrieved from the Planck function related to the sensors spectral response function. Surfaces to distinguish are for example evergreen forests and shrubs, deciduous forests and shrubs, sediments, inland water bodies, grasslands, bare soils or sand, urban areas or exposed rocks. Viewing angle might also have a small impact on emissivity, as mentioned by Wan and Dozier (1996). The results can be stored in a lookup table for further processing.

Furthermore, atmospheric factors are limiting factors of the model. These include uncertainties in molecular absorption, especially with respect to CWV, aerosol scattering and absorption coefficients, and uncertainties with atmospheric profiles. The MODTRAN model (Berk et al., 1987) was used by Wan and Dozier (1996) to model these atmospheric parameters. LST can only be derived under cloud-free conditions.

Bands 3-7, 13 and 16-19 are used to classify land cover and inferred LSE. Band 26 is used to detect cirrus clouds and TIR bands 20, 22, 23, 29 are used to correct for atmospheric effects. Bands 31 and 32 are used to derive LST itself. Atmospheric sounding channels can determine the temperature of the atmosphere and water vapor profiles. MIR bands are also used to correct solar radiation during the day and make it more distinguishable from thermal emission. In the following, the linear algorithm is briefly presented:

$$T_s = C + \left(A_1 + A_2 \frac{1-\varepsilon}{\varepsilon} + A_3 \frac{\Delta\varepsilon}{\varepsilon^2}\right) \frac{T_4 + T_5}{2} + \left(B_1 + B_2 \frac{1-\varepsilon}{\varepsilon} + B_3 \frac{\Delta\varepsilon}{\varepsilon^2}\right) \frac{T_4 + T_5}{2} \tag{6}$$

T_s is LST, ε is the emissivity, T_4 and T_5 are the radiance values of the two bands 31 and 32, and *A*, and

C are coefficients.

In this algorithm, the coefficients were determined using an extensive test data set and regression analysis. The atmospheric lower boundary temperature ranges from 256- 287K and CWV ranges from 0-2 cm. LST ranges also play a role in the regression. For simulation, subranges of the two ranges of T_s - T_{air} = -2 to +16K and -16 to +2K are created for the first iteration of the test data. The resulting LST value should subsequently determine which subrange should be used in the second iteration to produce the smallest possible errors. A similar procedure with subranges is used in the simulation for CWV in the interval of 0.5cm, as well as for the atmospheric lower boundary temperature.

The paper from Wan and Dozier (1996) was the main reference source of this chapter. In the name of readability, I have relegated the mentioning of this fact to this one note.

2.3 MODIS products

The two data products highlighted in grey in the table were used by the QGIS tool provided in this work. The products differ in their processing level from level 2 to level 3: whereas the level 2 processing represents finished LST values, without map projection yet while maintaining the original swath width, the level 3 processing provides both a projection (in the particular case of this work the MODIS sinusoidal projection) and image sections which no longer overlap, but rather have been arranged in tiles assigned a horizontal and a vertical number. Level 1 products have the same properties as Level 2 data, except for the data that is still unprocessed and recorded as radiance values. Other MODIS products are also displayed in degrees in an equal-angle geographic projection rather than metric. All products starting with MOD11 in the Earth Science Data Type (ESDT) name belong to the Terra satellite, whereas all MYD11 data-types belong to the Aqua satellite, with the same properties and product characteristics. (Wan, 2013)

MOD11_L2 is a full swath width product generated by the split-window algorithm (Zhengming Wan and Dozier, 1996). It has a 1km resolution and is shown in the geographic coordinate system with a projection referenced from its latitude and longitude. Here, the coordinates for five rows and five columns are also stored in the dataset.

The MOD11A1 product and the MOD11A2, which averages the MOD11A1 product for all eight days, with a resolution of 1km and 1200x1200 pixels, are a continuation of the MOD11_L2 product, also using the split-window algorithm described in section 2.2. Here, each tile represents one day in the sinusoidal projection.

The third product MOD11B1 is a tile according to the day/night algorithm (Zhengming Wan and Zhao-Liang Li, 1997) with a spatial resolution of 6km. The MOD11B2 product is equivalent to MOD11A2 an eight day average of the MOD11B1 product. The fifth product is a monthly variant of MOD11B1. The products ending with *C1, C2* and *C3*, which are represented in the geographic coordinate system and equal-area projection, are also the daily, eight-day, and monthly variants respectively and have a spatial resolution of 0.05°.

Earth Science Data Type (ESDT)	**Product Level**	**Nominal Data Array Dimensions**	**Spatial Resolution**	**Temporal Resolution**	**Map Projection**
MOD11_L2	L2	2030 or 2040 lines by 1354 pixels per line	1km at nadir	swath (scene)	None. (lat, lon referenced)
MOD11A1	**L3**	**1200 rows by 1200 columns**	**1km (actual 0.928km)**	**daily**	**Sinusoidal**
MOD11B1	L3	200 rows by 200 columns	6km (actual 5.568km)	daily	Sinusoidal
MOD11B2				eight-days	
MOD11B3				monthly	
MOD11A2	**L3**	**1200 rows by 1200 columns**	**1km (actual 0.928km)**	**eight days**	**Sinusoidal**
MOD11C1	L3	360° by 180° (global)	0.05° by 0.05°	daily	equal-angle geographic
MOD11C2	L3	360° by 180° (global)	0.05° by 0.05°	eight days	equal-angle geographic
MOD11C3	L3	360° by 180° (global)	0.05° by 0.05°	monthly	equal-angle geographic

Table 2: MODIS Land Surface Temperature Products available (Wan, 2013)

The scientific datasets (SDS) in the HDF used here can be seen in Table 3. The datasets for LST visualization are exclusively LST_Day_1km and LST_night_1km in this project. Quality control flags were not used, which makes the validity of the data less transparent.

The SDS are again subdivided into local attributes: In particular, the *_FillValue* (0) for all missing values, as well as calibration attributes such as the scale factor (0.02), which reduces the amount of data storage due to smaller decimal places, must be taken into account when reading out the data.

SDS Name	**Long Name**	**Number Type**	**Unit**	**Valid Range**	**Fill Value**	**scale factor**	**add offset**
LST_Day_1km	Daily daytime 1km grid Land-surface Tempera-ture	uint16	K	7500-65535	0	0.02	0.0
QC_Day	Quality control for day-time LST and emissivity	uint8	none	0-255	0	NA	NA
Day_view_time	(local solar) Time of day-time Land-surface Tem-perature observation	uint8	hrs	0-240	0	0.1	0
Day_view_angle	View zenith angle of day-time Land-surface Tem-perature	uint8	deg	0-130	255	1.0	-65.0
LST_Night_1km	Daily nighttime 1km grid Land-surface Tempera-ture	uint16	K	7500-65535	0	0.02	0.0
QC_Night	Quality control for nighttime LST and emis-sivity	uint8	none	0-255	0	NA	NA
Night_view_time	(local solar) Time of nighttime Land-surface Temperature observa-tion	uint8	hrs	0-240	0	0.1	0
Night_view_angle	View zenith angle of nighttime Land-surface Temperature	uint8	deg	0-130	255	1.0	-65.0
Emis_31	Band 31 emissivity	uint8	none	1-255	0	0.002	0.49
Emis_32	Band 32 emissivity	uint8	none	1-255	0	0.002	0.49
Clear_day_cov	day clear-sky coverage	uint16	none	0 - 65535	0	0.0005	0.
Clear_night_cov	night clear-sky coverage	uint16	none	0-65535	0	0.0005	0.

Table 3: The scientific datasets in the MOD11A1 product. (Wan, 2013)

2.4 Fields of application of MODIS data

The greatest advantage of MODIS LST in contrast to climate measurement data is the large-scale and uniform sensor coverage of the Earth. Based on this or in combination with other remote sensing data such as Land remote-sensing satellite (LANDSAT) or data from climate measuring stations, for example through the STARFM algorithm (Feng Gao et al., 2006), this information provides a basis for many fields of application.

Matson et al. (1978) already determined a higher temperature in urban than in rural areas with the help of satellite images. To date, Urban Heat Islands (UHI) are phenomena that have not yet been fully investigated. For the small-scale investigation of urban areas, especially from an urban planning perspective with regard to the greening of cities or the application of alternative building materials, scientific studies on UHI are still receiving attention today (Arnfield, 2003). UHI can be used, for example, for monitoring heat waves and their fatal consequences (Laaidi et al., 2012) or estimating soil moisture (Petropoulos et al., 2009).

In addition, LST from remote sensing data provides an important source for climate models of all kinds. They help to define more precisely input data such as net shortwave radiation for the calculation of evapotranspiration on agricultural land (Senay et al., 2007) or to build land surface energy balance models (Jin et al., 2007). In order to be able to study climate change and climatic global systems such as the surface temperatures of ice and snow of Arctic regions in relation to their temperature budget in detail, LST is already part of the fixed repertoire of the Earth Observation System (Cattle et al., 1995). In agriculture, it can also be used to determine the water requirements of wheat (Jackson et al., 1977), as well as frosts in orange groves or other frost-prone crops (Caselles and Sobrino, 1989). In addition, global climate models indicate that stronger summer monsoons are associated with higher land temperatures.

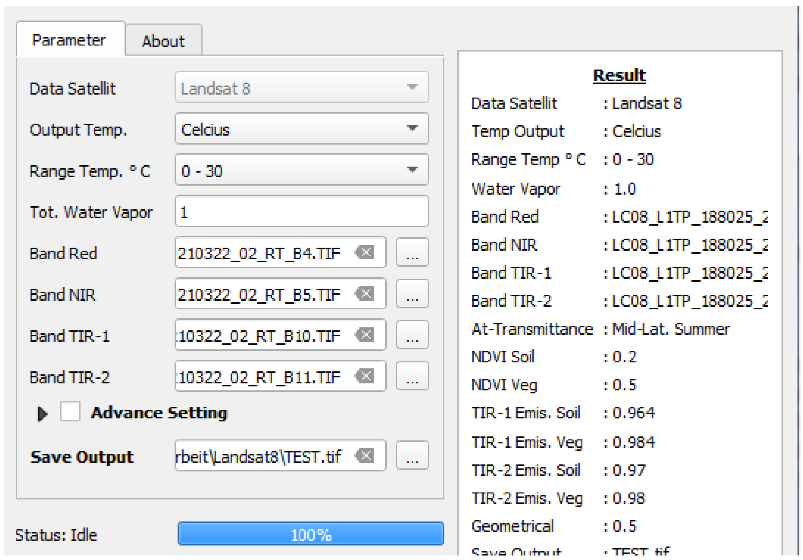
3. Comparable QGIS tools

In order to evaluate the usefulness of the developed QGIS plugin, similar tools are used for comparison. There is already a working Active Fire Plugin of great interest for the detection of fires with MODIS data, however no documentation is readily available. A QGIS MODIS LST plugin called MODIS_NITK (Hegde, 2019) also exists, but after many attempts it was not possible to install and use the tool properly. In the following two sections, only those plug-ins, of which this author could make direct use, will be explained in more detail. One is the LANDSAT8 SPT plug-in and the other one is the Land Surface Temperature plugin, which also uses LANDSAT data.

3.1 SPT Plug-In

Plug-In QGIS SPT is a tool that estimates LST using the split-window algorithm. It was developed by Muhammad Rahmahalim in Indonesia and has only been on the QGIS Plug-In Server since 2019 (Rahmahalim, 2020). Access to download can be found at https://plugins.qgis.org/plugins/qgis_spt .

This plug-in processes LANDSAT8 data and requires bands four (red), five (NIR), 10 (TIR1) and 11 (TIR2) as shown in Figure 3. The user can manually select the temperature units of the output °C, °F and °K, as well as the output path and name. The CWV value must also be entered manually, but as described in chapter 2.2, it can vary greatly locally with a range of 0.5 to 5.5. Consequently, this input parameter is rather inaccurate. While bands four and five are used to calculate the Normalized Difference Vegetation Index (NDVI) in order to classify the emissivity of the land surface, TIR bands 10 and 11 are used directly for the split-window algorithm according to Rozenstein et al. (2014). The data must be downloaded manually beforehand, for example in the USGS Earth Explorer. However, only soil and vegetation are distinguished as emission values with the values 0.964 and 0.984 for TIR1, and 0.970 and 0.980 for TIR2. For atmospheric transmission, two standard profiles for mid-latitudes in summer and the 1976 US standard were used. Conditions and The NDVI classification values, the atmospheric transmittance, and the emission values can also be adjusted manually. It is also worth noting that quality flags and cloud masks are missing in this plug-in. Compared to MODIS LST, the results are less



Data Satellit	Landsat 8
Output Temp.	Celcius
Range Temp. ° C	0 - 30
Tot. Water Vapor	1
Band Red	210322_02_RT_B4.TIF
Band NIR	210322_02_RT_B5.TIF
Band TIR-1	10322_02_RT_B10.TIF
Band TIR-2	10322_02_RT_B11.TIF
Save Output	rbeit\Landsat8\TEST.tif

Status: Idle 100%

Result

```
Data Satellit       : Landsat 8
Temp Output         : Celcius
Range Temp ° C      : 0 - 30
Water Vapor         : 1.0
Band Red            : LC08_L1TP_188025_2
Band NIR            : LC08_L1TP_188025_2
Band TIR-1          : LC08_L1TP_188025_2
Band TIR-2          : LC08_L1TP_188025_2
At-Transmittance    : Mid-Lat. Summer
NDVI Soil           : 0.2
NDVI Veg            : 0.5
TIR-1 Emis. Soil    : 0.964
TIR-1 Emis. Veg     : 0.984
TIR-2 Emis. Soil    : 0.97
TIR-2 Emis. Veg     : 0.98
Geometrical         : 0.5
Save Output         : TEST.tif
```

Figure 3: User Interface of SPT Plugin

accurate due to the small number of surface and atmospheric parameters, but this tool has the potential in combination with MODIS data to achieve a higher spatial resolution of 100m in a recurring interval of 16 days from the launch of the LANDSAT8 satellite on 02-11-2013 onward.

3.2 Land Surface Temperature Estimation Plugin

This plugin according to Isaya Ndossi and Avdan (2016) is designed to extract LST from TIR images of LANDSAT5, LANDSAT7 and LANDSAT8. It also allows to convert LANDSAT digital numbers to radiance, radiance to brightness temperature and finally to calculate LSE. Unfortunately, this plugin can only be used in QGIS2 and can be downloaded from https://plugins.qgis.org/plugins/LandSurfaceTemperature .

An interesting feature of this tool is the subdivision of the individual sections, which on the one hand makes the procedure more comprehensible and preserves the intermediate results, while on the other hand proving to be potentially excessively time consuming. First, the sensor type must be selected. Depending on which sensor is selected, a single band can be chosen. If LANDSAT8 is used as input, the user must manually enter an offset calibration factor of -0.29 according to the documentation for LANDSAT8. Again, data must be downloaded in advance. A thermal band and the metadata text file must then be inserted via the browse buttons. The calculated result can then be calculated to Brightness Temperature in the next tab with a similar input pattern. In the next step, the NDVI is calculated using the red and NIR bands, from which the LSE is calculated following one of two methods: One is NDVI based on (Zhang et al., 2006) which expresses the vegetation as NDVI, as well as classifying the emissivity as shown in Table 4.

NDVI	**LSE**
NDVI < -0.185	0.995
0.1855 <= NDVI < 0.157	0.985
0.157 <= NDVI <= 0.727	1.009 + 0.047 x ln(NDVI)
NDVI > 0.727	0.990

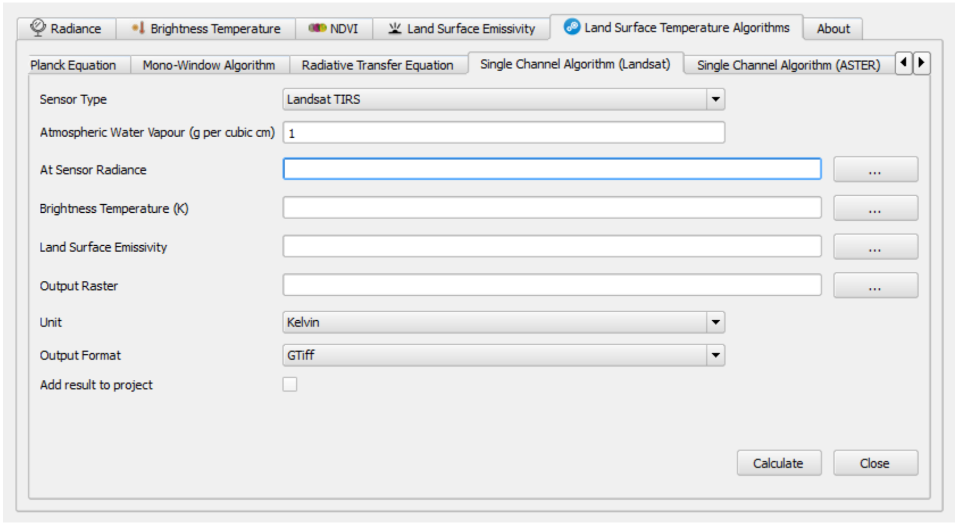
Table 4: NDVI classification parameters for LSE (Isaya Ndossi and Avdan, 2016)

The NDVI threshold emissivity distinguishes between soil and vegetation pixels, which are normally classified at NDVI values 0.2 and 0.5. Using these values, the land surface is divided into vegetation classes, which also have a specific emissivity. If this classification is not successful because a pixel consists of several surfaces, the so-called fractional vegetation cover is used, which weighs the emissivity proportionally.

The LST can be derived from this information in the last tab. A total of four different methods can be found here: The first is the Planck function converted to *T*; The second is the single channel algorithm according to Jiménez-Muñoz and Sobrino (2003), which was already briefly described in chapter 2.2; The third is the radiative transfer model, described in chapter 2.1.3.

The fourth method, the mono-window algorithm according to (Liu and Zhang, 2011) is very closely related to the single channel algorithm. The atmospheric profiles used here were USA 1976 and Mid-latitude summer as in the algorithm described in chapter 3.1, as well as the Tropical model and Mid-latitude winter model. The profiles used depend on the study area and the date. The details can be found in the publication mentioned above or in the publication of the plugin by Isaya Ndossi and Avdan (2016).

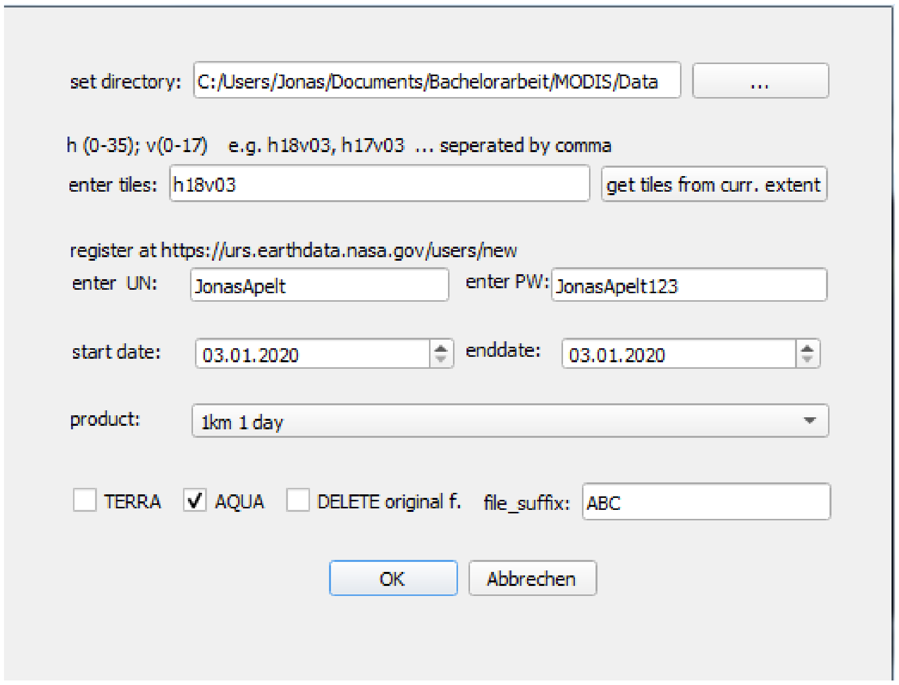
Figure 4 shows the user interface in the LST Algorithms tab in the Single Channel Algorithm (Landsat) column. Similar to the previous section, the plugin also uses LANDSAT data, which is only available every 16 days, but at a higher resolution. The Land Surface Temperature Estimation Plugin would also be a good addition to the MODIS LST Download and Processing QGIS Plugin developed in this thesis, although some disadvantageous elements might be the manual data download and it's, at times, difficult handling. In addition, the MODIS algorithm is much more precise than the algorithms presented here. (Wan, 1999)


Radiance
Brightness Temperature
NDVI
Land Surface Emissivity
Land Surface Temperature Algorithms
About
Planck Equation
Mono-Window Algorithm
Radiative Transfer Equation
Single Channel Algorithm (Landsat)
Single Channel Algorithm (ASTER)
...
Calculate
Close

Sensor Type	Landsat TIRS
Atmospheric Water Vapour (g per cubic cm)	1
At Sensor Radiance	
Brightness Temperature (K)	
Land Surface Emissivity	
Output Raster	
Unit	Kelvin
Output Format	GTiff
Add result to project	☐

Figure 4: User Interface of Land Surface Temperature Plugin

4. MODIS LST Download and Processing QGIS Plugin


set directory: C:/Users/Jonas/Documents/Bachelorarbeit/MODIS/Data

h (0-35); v(0-17) e.g. h18v03, h17v03 ... seperated by comma

enter tiles: h18v03

register at https://urs.earthdata.nasa.gov/users/new

enter UN: JonasApelt

enter PW: JonasApelt123

start date: 03.01.2020

enddate: 03.01.2020

product: 1km 1 day

☐ TERRA ☑ AQUA ☐ DELETE original f.

file_suffix: ABC

Figure 5: UI of MODIS LST Download and Processing Plugin

MODIS LST download and Processing Plug-In is an extension to the open source platform QGIS. It was developed in the programming language Python, which is supported by the Application User Interface (API) in QGIS. The framework of the application, i.e. the references to the objects of the user interface (UI), was built in PyQt5. The processing of the data was done with in *gdal* from osgeo library. The download of the MODIS data is supported by the *pymodis* module ("About pyModis — pyModis 2.1.0 documentation," 2021.).

The plug-in allows access to public MODIS LST data, which can otherwise only be retrieved manually. First, LST data is downloaded from the USGS server in the selected time frame. This eliminates a potential source of human error, for example by incorrectly selecting the data set. It converts the original HDF data, which cannot be read directly in QGIS, into the Tagged Image File Format (TIFF) and additionally outputs a day and night average as well as an average of all files of the entire time span. To validate the data, three additional rasters are created for the respective mean value raster. They count the proportion of pixels that do not contain NAN-values on the total number of the merged time scale. Since cloud visibility is subject variation from day to day, a longitudinal slice over a longer period of time can produce mean value images with almost zero missing values. However, some parts of the generated mean are missing more often than other parts, e.g. due to climatic conditions. This correlation is shown in the *COUNT_VALUE* file.

In the top line of the tool, the storage path of the created files is inserted first. If the user does not want to enter the path manually, a folder path can be selected by clicking on the button next to it. In this folder, depending on the selection of the product type in the combo box “*Product*", either a *oneday*-folder for the *MOD11A1* data and the selection in the UI "*1km 1day*" or, if the "1km 8day" option is selected, corresponding to the *MOD11A2* product, a folder *8day_mean* can be created. In the respective folder, two further folders are created, one for the original data (unmerged tiles) and one for the merged rasters (merged tiles).

Under the selection of the folder path, the desired tiles of the MODIS raster can be entered in a *lineEdit*, as can be seen in Figure 5. The specific names are entered as text, separated by a comma, but can also be inserted automatically over the current image section using the button next to it. Below this, the user name and password must be entered manually in the line provided for this purpose. By default, the access data is provided by the developer. However, an account can also be created at https://e4ftl01.cr.usgs.gov . There you will find the database that *pymodis* can access. In the window below, the desired period must be selected. *01.01.2020* is entered as the default. Time spans from one day to any size can be selected. It is worth noting that a more expansive variable range (i.e. the number of tiles selected, the time span, the product type etc.), will inevitably result in a lengthier processing time. In the last section, three fields can be clicked. If you select the first field "*TERRA*", all data of the Terra mission will be loaded, if you select the field "*AQUA*"; all data of the Aqua mission will be loaded. The most accurate results are obtained by selecting both satellites. In the third field, the user can decide whether the HDF files from the download and the unmerged day/night TIFF files should be deleted or not. In some cases, very large amounts of data are generated, thus adding to the appeal of this tool.

In order to avoid undesired results in case of multiple executions of the application, e.g. files from a previous output being taken over into the result of a new application execution, an extension of the file name for the merged files can be defined in the last *lineEdit* "*file_suffix*". This option, as well as deleting the output data, is strongly recommended for quality assurance.

4.1 Workflow and problems

The work process as a whole consists of numerous failed attempts. Each section was tested as an example and other attempts were discarded again. It should not be discussed in too much detail here, but a few systematic problems that occurred should be mentioned.

The plugin was created in Windows and accordingly only runs in Windows. A core problem, which could also occur with other users, is the assignment of a new environment variable *PROJ_LIB* for the current user account. *gdal* must access a file called *proj.db*. Therefore, the path to this file must be stored under this new environment variable. Before the code was structured, no functions were created. Only in the later course of the work were recurring elements summarized, thus making it much easier to read.

Another problem for a long time was the lack of the possibility to download a single day. Overall, the documentation of the *pymodis* library is very scarce. The problem could be solved by the function *downloadsAllDay()* and the parameter *allDays = False*.

The library *rasterio* offers numerous, easy-to-use possibilities for spatial data processing in Python. Unfortunately, QGIS cannot work with this package, which is why the raster processing was done with *gdal*.

Finally, it should be pointed out that output raster layers cannot be directly inserted into QGIS automatically. The function *addRasterLayer()* from *qgis.utils* module *iface* gives an error that the TIFF is not supported.

To test the applications functionality it was run multiple times on three other devices. During this procedure a problem occurred when the username in windows contained white spaces. To circumvent this, the download folder should than be chosen differently. In addition when starting the plugin, the map canvas should get projected automatically to the MODIS projection. This only works if a standard map was chosen and loaded in QGIS before the Plugin is started.

4.2 Design and structure

The plugin was created with *Pluginbuilder3*, an extension in QGIS. It defines the framework of the application. This implements the packages *qgis.PyQt*, *qgis.utils* and *qgis.core*. All other extensions are also imported here. We used *datetime*, *os*, *pandas, requests, pathlib, imp, subprocess, downmodis* from *pymodis*, *numpy, osr* and *gdal* from *osgeo* library, *glob* and *getpass*. The code described is contained as *MODIS.py* in the plugin folder.

Figure 7 shows the flowchart of the application structure, for a better understanding of the descriptions in this chapter. Automatically created by the plugin builder are the first functions from line 59 to line 200. The first function *__init__()* initialises the folder path of the plugin and connects the files inside with the QGIS interface. Then the plugin name is specified in line 85. The following function *tr()* translates strings into Qstrings in order to display them later in QGIS. With the following function *add_action()* an icon can be added to the toolbar. The exact specifications and adjustments to the icon are well documented in the comments of the code. The core function of the Graphical UI is the *InitGUI()*- function. In this case, only the icon, toolbar text with the name of the plugin, the command to run the program in the main QGIS window and the callback function that performs the actual raster processing as the *run()* function are included. With unload, the added icon and the menu can also be removed again, which was omitted here. However, the function remains in the code to facilitate later customization.

The function *select_output_file()* is responsible for selecting the working directory. *QFileDialog.getExistingDirectory()* from *qgis.PyQt.QtWidgets* opens a window in the current operating system that lets you select a folder path and saves it as a string. This string is temporarily stored in the *lineEdit* (208) created next to it.

In order to be able to download the correct image sections, the user must enter the right tile name in the case of manual input, as illustrated in Figure 6. To simplify the handling, the current map section should directly display the correct tile names. In order to first establish the proper coordinate reference, the map is transferred to the MODIS projection as standard in line 283. First the coordinates of the map section must be determined. *Iface()* from *qgis.utils* can read out the X and Y coordinates with the method *extent*. To compare this initial information, a table with

all tile names and the corresponding coordinates is needed. The tutorial is from ("Sinusoi-dalMODIS – Marine Geospatial Ecology Tools," n.d.) website. Here you will also find a detailed explanation of the MODIS projection. The basic assumptions are on the one hand the origin co-ordinate of the lower left corner with the values x = -20015109.354m and y= -10007554.677m, on the other hand the height and width of the tiles with 1111950.5196666666m each. From this coordinate origin, all other points in the MODIS sinusoidal grid can be easily calculated and saved in a table. The names of the tiles were downloaded from the NASA server using *requests* at https://modis-land.gsfc.nasa.gov/pdf/sn_bound_10deg.txt . On this basis, the correct row of the table can be selected with a query in *pandas*. To ensure that the query is not empty if a tile is larger than the entire map section, the tile height is subtracted from the min values and added to the max values so that at least one tile is always selected (269-272). The selection made is then saved in the *lineEdit_2* field, where tile names can also be entered manually.
From line 279 the *run*() function begins, which does the "real" work of the application. At the beginning, the buttons are activated on the condition that the application has not yet been executed. Both the button for the memory path (291) and the button for the tile names (293) are activated here by clicking.

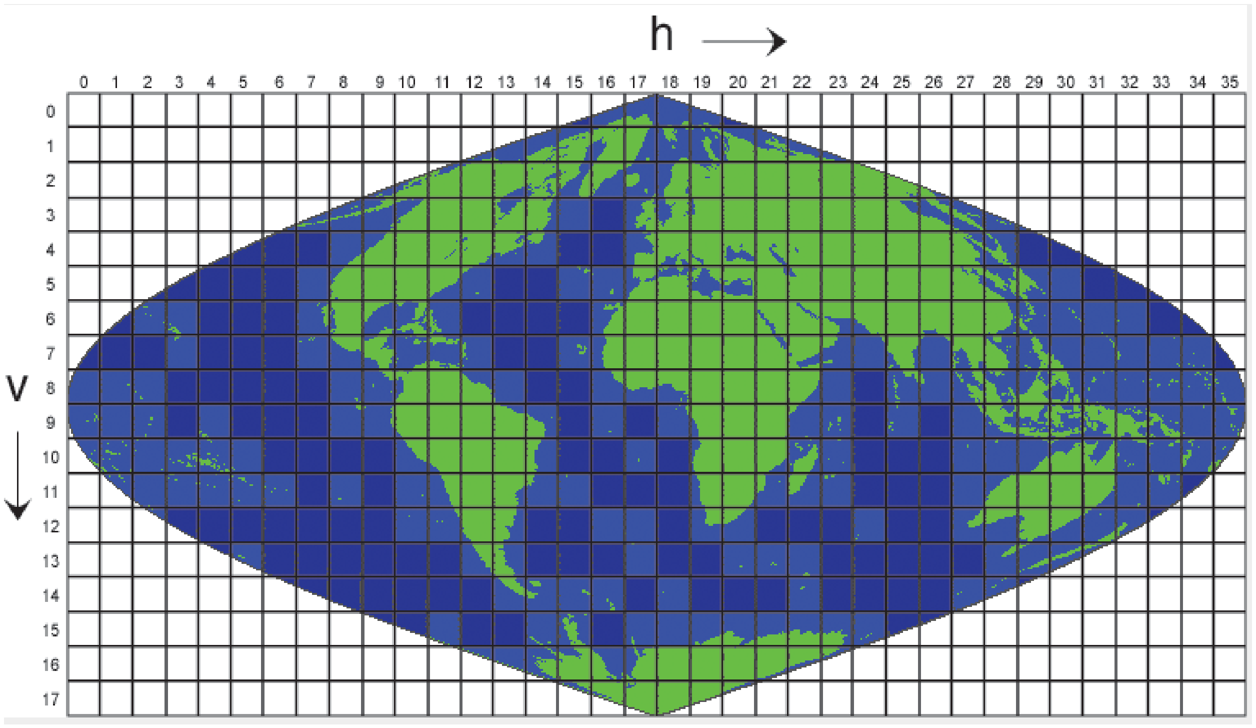


Figure 6: MODIS TilesCoordinate System in sinusoidal projection

Then the *ComboBox* for the product selection is populated with the "*1km 1 day*" and "*1km 8 days*" strings (298). These are used instead of the actual designation, e.g. MOD11A2, to increase comprehensibility for the user. The other *lineEdits* are also provided with the default values already mentioned. When the "*OK*" button is pressed, the calculation process is started.

The tiles previously stored in the *lineEdits* of the user interface, the suffix and the memory path are read and stored as variables (313-315; 319-323). If the string with tiles was not entered correctly as described above, a correction is made by deleting superfluous spaces and separating individual tiles with commas (316-317). In lines 326 to 332, the start and end dates are stored as variables of the respective *dateEdit*. *today* is always at a later time or at the same time as *enddate*. If *today* is earlier than *enddate*, the dates after *enddate* and before *today* are selected. The elements previously stored in the *comboBox* must now be transferred to the original product designation (335-339). With a simple if-condition, the second variable part of the product key is stored. This means that the beginning of the string *MOD11A1.006* or *MOD11A2.006*, namely *MOD11* is omitted and the second part *A1.006* or *A2.006* is stored. Lines 343 to 356 check whether the *checkBox* is selected and output this information as a TRUE or FALSE value in a variable, which was done for clarity and simplification in the code.

Depending on the product type, a new folder is created with *Path()* from *pathlib* and the working directory is adapted accordingly (365-373).

In the following, the basic functions are created with which the main loops will later run. The first function is the *download* function, which uses the *downmodis()* function from the *pymodis* package and has been specially adapted for this purpose (376-383). The exact documentation can be found at www.pymodis.org/info.html . Subsequently you will find the *get_data()* function, which can read out the day and night surface temperatures with *gdal* (386-400). Sub-data sets are read from the main data set as shown in Table 3. The numbers for *LST_Day_1km* and *LST_Night_1km* are 0 and 4. The data set is finally read out as a *numpy* array and multiplied by the *scale_factor* 0.02. This is used to save memory. In addition, all *FillValues = 0* are converted to *np.nan*. So that the array can later be converted as a TIFF in the function *to_tiff()* (403-416), the variable *band_path* and for georeferencing *band_ds* are also stored in the environment. This function works exclusively with the *gdal* library. *GetDriverByName()* can generate a TIFF file with a file path, the raster dimensions (Table 2) of the projection, as Well-Known-Text Format (WKT) and the array, which can also be visualized as a QGIS layer. The projection text can be copied at *spatialreference.org.*

The *delete_down()* function (419-433) can delete all data that is no longer needed. To do this, a list of all files in the folder *unmergedTiles* is created and all files ending with *".hdf", ".xml", ".txt", "log", "day.tiff* and *"night.tiff"* are deleted. A similar procedure is followed in the folder *mergedTiles*, where only the endings *"day.tiff* and *"night.tiff"* are deleted. The iteration via *listdir* was carried out with *try* and *except* method, since errors occurred again and again when deleting individual files. Thus, this problem could be circumvented. Presumably, the error occurs when certain files have not yet been closed, but this could not be solved.

In order to be able to read data in *get_data()* first, it needs a list of all HDF files *listFile*. The *getFileList()* function was designed for this purpose (436-455). It was decided to do without if-conditions and to work with *try* and *except* instead, which simplifies the function considerably. Fortunately, a text file of a list of files is already created during the download. Therefore, this list must be read for both product types and for both satellites. For this purpose, an attempt is made to create two separate lists, one for Terra and one for Aqua. These two lists are merged into one. To distinguish the product, only the number is inserted as *num*, i.e. "*1*" or "*2*" in the file path of *listfileMOD11A'+ num +'.006.txt'*. If an exception occurs, an empty list is created.

Depending on the input, tiles can have different data types. In order for the download function to read the strings correctly and to be able to run a loop over the tiles later, they should ideally be available as a list. For this purpose, the data type of the input is adapted in lines 461 to 486. Now *get_data()* and *to_tiff()* are executed twice, once for night data and once for day data in a loop for each tile (499- 504).

The next section merges all tiles with the same name if they are from the same day, the same satellite and the same time of day. This creates a large scene of a snapshot, depending on the amount of tiles selected. For this purpose, a new folder is created at the same level as *unmergedTiles*, *mergedTiles* (513-516). With *glob(),* a list of all TIFF files can be created. From this list, the string before the tile name and after the tile name is saved, both are merged and then the duplicate values are eliminated with *numpy.unique()*, resulting in a list of unique tiles (522-

524). The merge function is executed by a script from the *gdal* library called *gdal_merge.py*. Because this script could not be found in the QGIS environment, it was placed in the plugin folder and the current working directory was adapted to the plugin path (537). To find the path of the plugin individually, the username must be saved with *getpass.getuser()* (528), and then all file names in this folder must be obtained with *os.walk()*. After that search for *MODIS.py*, because it is assumed that this file is stored once in the same place as gdal_merge.py (530-535). If the correct path was found, *gdal_merge.py* should now run properly. Finally, using the unique results *res*, a list of all duplicate files, apart from the tile itself, is formed for each element in *res*, which can then be passed into a single long string using *subprocess()*, which executes command line code of *gdal_merge.py* (539-543).

In the last part, the average values are formed in the longitudinal section for night, day and total value and their corresponding indication of missing values under the condition that only *MOD11A1* is available as a product type (601). For this purpose, two functions are defined in advance. The first function *mean()* builds a stack of all arrays of a list of file paths in line 559 to 563. With *numpy.nanmean()*, the mean value of all arrays in the stack can be calculated, disregarding the NAN values (565). Then the new array is converted into a TIFF as in to_tiff() and this time with the user-created suffix (567-579). A similar procedure is followed for the missing values, for which all values greater than zero (and thus not NAN) are set equal to 1 in the *sum_tiff()* function (583). With *numpy.nansum()* all values can be added, disregarding the missing values, which in the result means the number of existing values per pixel. Divided by the number of all files and multiplied by 100, the result is a percentage value of the existing values per pixel (584). This new array can now be written as a TIFF (588-599). The lists mentioned at the beginning, which are needed as input in both functions, are created for all TIFF files (605), for all TIFF files with night values (606) and for all TIFF files with day values (607). For each of these three lists, the mean function is executed once (610-614). The *list_stacks* that are also saved can be used in *sum_tiff()* (648-650). In lines 617 to 639, a further mean value is formed from the mean values of day and night, which should provide a more accurate result than the mean value from all arrays, because this avoids a possible imbalance of missing night or day values. This means that if, for example, night values occur much less frequently than day values, they are underrepresented in the mean value from all arrays and the result is distorted. With an average of day and night data, this distortion is avoided.

Finally, the *delete_down()* function is applied to the *MOD11A1* products (657-658) if the selection has been activated by the user. For *MOD11A2 delete_down()* cannot be used because only files from *unmergedTiles* folder need to be deleted (658-666). When all actions are completed a "Success" status message appears in QGIS (683-686).

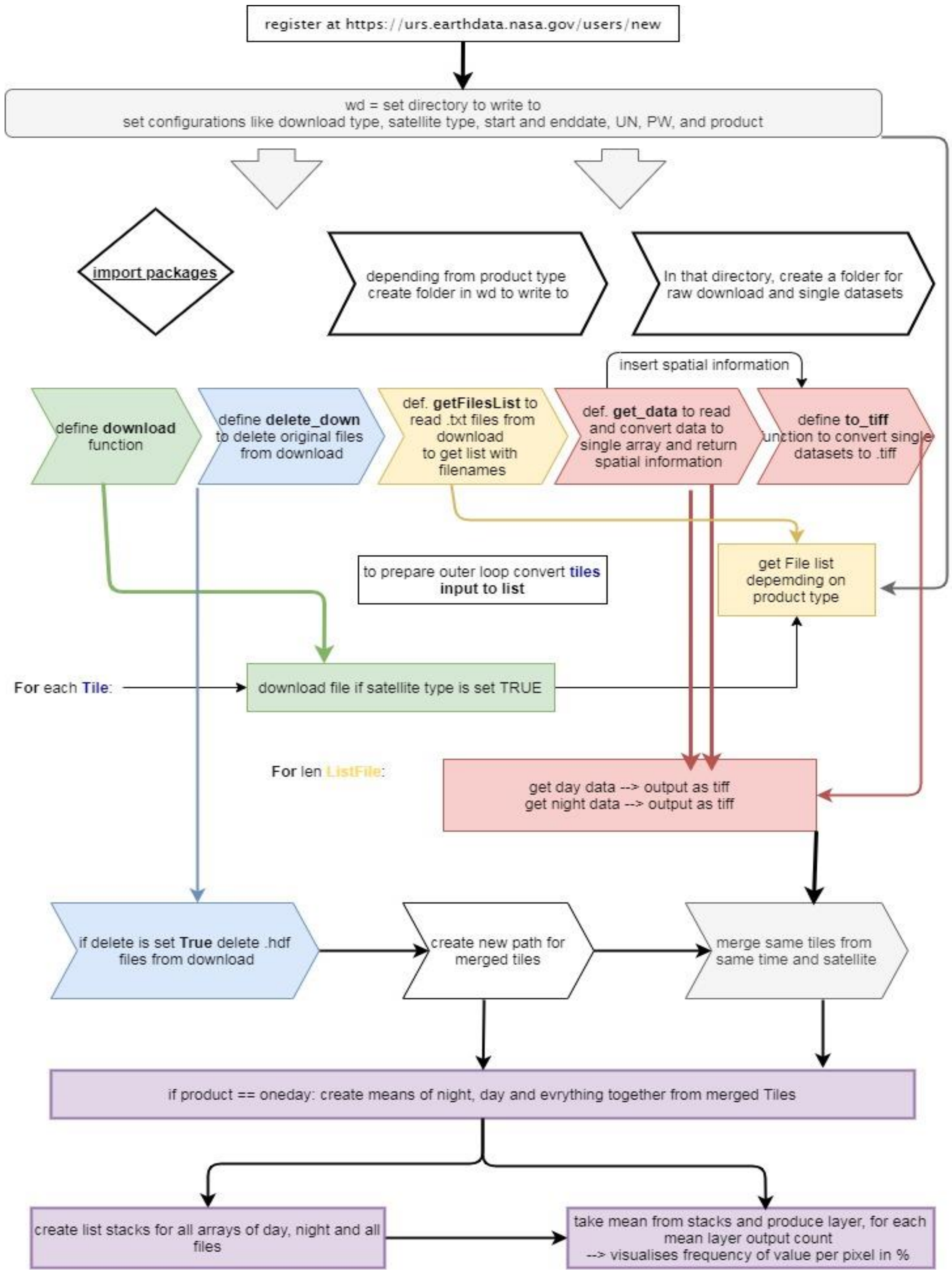

register at https://urs.earthdata.nasa.gov/users/new
wd = set directory to write to
set configurations like download type, satellite type, start and enddate, UN, PW, and product
import packages
depending from product type create folder in wd to write to
In that directory, create a folder for raw download and single datasets
insert spatial information
define download function
define delete_down to delete original files from download
def. getFilesList to read .txt files from download to get list with filenames
def. get_data to read and convert data to single array and return spatial information
define to_tiff function to convert single datasets to .tiff
get File list depemding on product type
to prepare outer loop convert tiles input to list
For each Tile:
download file if satellite type is set TRUE
For len ListFile:
get day data --> output as tiff
get night data --> output as tiff
if delete is set True delete .hdf files from download
create new path for merged tiles
merge same tiles from same time and satellite
if product == oneday: create means of night, day and evrything together from merged Tiles
create list stacks for all arrays of day, night and all files
take mean from stacks and produce layer, for each mean layer output count
--> visualises frequency of value per pixel in %

Figure 7: Flowchart

5. Conclusion

In this work, a QGIS plugin was developed that can download, process and visualize LST data. Compared to other existing applications, it is a useful and unique extension of existing plugins. The clear advantage is given by the direct download, while other plugins require a manual download. The only disadvantage is the longer processing times. The developed application is the only functioning one of its kind.
However, its development is not quite finished. Possible errors cannot be ruled out due to the insufficient frequency of test runs. Suggestions for improvement lie above all in the intersection with other data sets such as LANDSAT, land use or climate measurement data. Especially the low resolution of 1km pixels makes an application for the detection of UHI difficult. The validation of the data has already been carried out many times. (Wan, 2014). The LST data from the MODIS instrument is a very accurate model of reality with maximum deviations <1*K*. (Wan, 2014). The temporal resolution with a total of four images from two satellites per tile and per day is also unique. Unfortunately, it was not possible to provide more detailed information on the functioning and coefficients of the algorithm, as the data material produced by the author is only available on request and no information has been provided so far.

6. Commented Code

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 MODIS
                                 A QGIS plugin
 This plugin is made to download and visualise MODIS LST data. you can get
raw data from the usgs server, select multiple tiles, get mean values of a
timespan and download different product types such as 1km-1day or 1km-8day
product and MODIS AQUA or MODIS Terra.
 Generated by Plugin Builder: http://g-sherman.github.io/Qgis-Plugin-Builder/
                             -------------------
        begin                : 2021-02-18
        git sha              : $Format:%H$
        copyright            : (C) 2021 by Jonas Apelt
        email                : jonas.apelt@gmx.de
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
"""
from qgis.PyQt.QtCore import QSettings, QTranslator, QCoreApplication
from qgis.PyQt.QtGui import QIcon
from qgis.PyQt.QtWidgets import QAction, QFileDialog
from qgis.core import QgsProject, Qgis, QgsRasterLayer, QgsApplication,
QgsCoordinateReferenceSystem
import datetime
# Initialize Qt resources from file resources.py
from .resources import *
# Import the code for the dialog
from .MODIS_dialog import MODISDialog
import pandas as pd
import requests
from qgis.utils import iface
from pathlib import Path
import imp
import subprocess
packages = ['os', 'osgeo', 'pymodis', 'glob', 'pathlib', 'getpass']
for i in packages:
    try:
        imp.find_module(i)
        pass
    except ImportError:
        subprocess.check_call(['python', '-m', 'pip', 'install', i])

from pymodis import downmodis
import os # This is is needed in the pyqgis console also
import os.path
import numpy as np
from osgeo import gdal, osr
import glob
import getpass


class MODIS:
    """QGIS Plugin Implementation."""

    def __init__(self, iface):
        """Constructor.

        :param iface: An interface instance that will be passed to this class
            which provides the hook by which you can manipulate the QGIS
            application at run time.
        :type iface: QgsInterface
        """
        # Save reference to the QGIS interface
        self.iface = iface
        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'MODIS_{}.qm'.format(locale))

        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)
            QCoreApplication.installTranslator(self.translator)

        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&MODIS LST download and processing')

        # Check if plugin was started the first time in current QGIS session
        # Must be set in initGui() to survive plugin reloads
        self.first_start = None

    # noinspection PyMethodMayBeStatic
    def tr(self, message):
        """Get the translation for a string using Qt translation API.

        We implement this ourselves since we do not inherit QObject.

        :param message: String for translation.
        :type message: str, QString

        :returns: Translated version of message.
        :rtype: QString
        """
        # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
        return QCoreApplication.translate('MODIS', message)


    def add_action(
        self,
        icon_path,
        text,
        callback,
        enabled_flag=True,
        add_to_menu=True,
        add_to_toolbar=True,
        status_tip=None,
        whats_this=None,
        parent=None):
        """Add a toolbar icon to the toolbar.

        :param icon_path: Path to the icon for this action. Can be a resource
            path (e.g. ':/plugins/foo/bar.png') or a normal file system path.
        :type icon_path: str

        :param text: Text that should be shown in menu items for this action.
        :type text: str

        :param callback: Function to be called when the action is triggered.
        :type callback: function

        :param enabled_flag: A flag indicating if the action should be ena-
bled
```

```
            by default. Defaults to True.
        :type enabled_flag: bool

        :param add_to_menu: Flag indicating whether the action should also
            be added to the menu. Defaults to True.
        :type add_to_menu: bool

        :param add_to_toolbar: Flag indicating whether the action should also
            be added to the toolbar. Defaults to True.
        :type add_to_toolbar: bool

        :param status_tip: Optional text to show in a popup when mouse
pointer
            hovers over the action.
        :type status_tip: str

        :param parent: Parent widget for the new action. Defaults None.
        :type parent: QWidget

        :param whats_this: Optional text to show in the status bar when the
            mouse pointer hovers over the action.

        :returns: The action that was created. Note that the action is also
            added to self.actions list.
        :rtype: QAction
        """

        icon = QIcon(icon_path)
        action = QAction(icon, text, parent)
        action.triggered.connect(callback)
        action.setEnabled(enabled_flag)

        if status_tip is not None:
            action.setStatusTip(status_tip)

        if whats_this is not None:
            action.setWhatsThis(whats_this)

        if add_to_toolbar:
            # Adds plugin icon to Plugins toolbar
            self.iface.addToolBarIcon(action)

        if add_to_menu:
            self.iface.addPluginToMenu(
                self.menu,
                action)

        self.actions.append(action)

        return action

    def initGui(self):
        """Create the menu entries and toolbar icons inside the QGIS GUI."""

        icon_path = ':/plugins/MODIS/icon.png'
        self.add_action(
            icon_path,
            text=self.tr(u'MODIS download and visualization'),
            callback=self.run,
            parent=self.iface.mainWindow())

        # will be set False in run()
        self.first_start = True


    def unload(self):
        for action in self.actions:
            self.iface.removePluginMenu(
                self.tr(u'&MODIS LST download and processing'),
                action)
            self.iface.removeToolBarIcon(action)

    # define function to get directory path to write to by Buttonclick
    def select_output_file(self):
        # open a window to choose path
      wd = QFileDialog.getExistingDirectory(
        self.dlg, "Select output directory ")
      #put pathstring in lineEdit
      self.dlg.lineEdit.setText(wd)

        #define function to get tilenames from extent on Buttonclick
    def tilesFromExtent (self):
        #https://code.env.duke.edu/projects/mget/wiki/SinusoidalMODISimport
requests
        #get current extent of map
        extent = iface.mapCanvas().extent()
        ################################################
        #find tile, which is the closest to that extent!
        ################################################
        #get coordinates of current map
        xmax = extent.xMaximum()
        ymax = extent.yMaximum()
        xmin = extent.xMinimum()
        ymin = extent.yMinimum()

        ##########################
        ##      parameters      ##
        ##########################
        tile_width = 1111950.5196666666
        tile_height = 1111950.5196666666
        d = tile_width

        # toto : down left corner of the coordinate system
        toto = -20015109.354
        tutu = -10007554.677
        ##########################

        # use download to get column with tile names
        direc = Path().cwd()
        url = 'https://modis-land.gsfc.nasa.gov/pdf/sn_bound_10deg.txt'
        tiles_txt = requests.get(url, allow_redirects=True)

        tile_bounds = direc.joinpath('tiles_bounds.txt')
        tile_bounds.write_bytes(tiles_txt.content)
        # always open files with a with statement to avoid unclosed files
        with tile_bounds.open() as f:
            text = f.read()
            text = text.splitlines()

            del text[0:7]
            del text[-2:]

        # the file is now useless unlink it
        tile_bounds.unlink()

        df_bounds = pd.DataFrame(columns=['tile','horizontal', 'verti-
cal','xmin_m','xmax_m', 'ymin_m', 'ymax_m'])

        hor, ver, xmin_m, xmax_m, ymin_m, ymax_m=[], [], [], [],[], []

        # create a dataframe and calculate bounds of tiles
        df_bounds.tile = [f"h{text[i][5:7].replace(' ',
'0')}v{text[i][1:3].replace(' ', '0')}" for i in range(len(text))]
        df_bounds.horizontal = df_bounds.tile.map(lambda row:
row[1:3]).astype(int)
        df_bounds.vertical = df_bounds.tile.map(lambda row:
row[4:6]).astype(int)
        df_bounds.xmin_m = df_bounds.horizontal.map(lambda r: toto +
r*tile_width)
        df_bounds.ymin_m = df_bounds.vertical.map(lambda r: tutu + (17 -
r)*tile_height)
        df_bounds.xmax_m = df_bounds.xmin_m.map(lambda r: r + tile_width)
        df_bounds.ymax_m = df_bounds.ymin_m.map(lambda r: r + tile_height)

        #filter the names of the tiles inside the box
        tile_names = df_bounds[(df_bounds['xmin_m'] >= xmin-d) &
                               (df_bounds['xmax_m'] <= xmax+d) &
                               (df_bounds['ymin_m'] >= ymin-d) &
                               (df_bounds['ymax_m'] <= ymax+d)]

        # improvement for the print
        tile_names = ','.join(tile_names.tile)
        self.dlg.lineEdit_2.setText(tile_names)


    def run(self):
        """Run method that performs all the real work"""
        #when starting thge plugin change the CRS to target MODIS crs
        crs_wkt = 'PROJCS["unnamed",GEOGCS["Unknown datum based upon the cus-
tom spheroid",DATUM["Not_specified_based_on_custom_spheroid",SPHEROID["Custom
spheroid",6371007.181,0]],PRIMEM["Greenwich",0],UNIT["de-
gree",0.0174532925199433,AUTHORITY["EPSG","9122"]]],PROJECTION["Sinusoi-
dal"],PARAMETER["longitude_of_center",0],PARAMETER["false_easting",0],PARAME-
TER["false_northing",0],UNIT["Meter",1],AXIS["Easting",EAST],AXIS["North-
ing",NORTH]]'
        self.iface.mapCanvas().setDestinationCrs(QgsCoordinateReferenceSys-
tem(crs_wkt))

        # Create the dialog with elements (after translation) and keep refer-
ence
```

```
        # Only create GUI ONCE in callback, so that it will only load when
    the plugin is started
        if self.first_start == True:
            self.first_start = False
            self.dlg = MODISDialog()
            #activate Button to select output directory
            self.dlg.pushButton.clicked.connect(self.select_output_file)
            #activate Button to get string of tiles from current map extent
            self.dlg.pushButton_ext.clicked.connect(self.tilesFromExtent)

        items = ['1km 1 day', '1km 8 days']
        self.dlg.comboBox.clear()
        # Populate the comboBox with names of all the loaded layers
        self.dlg.comboBox.addItems([it for it in items])
        password="JonasApelt123"
        user="JonasApelt"
        # populate user and PW editLines with default values
        self.dlg.lineEdit_UN.setText(user)
        self.dlg.lineEdit_PW.setText(password)
        # show the dialog
        self.dlg.show()
        # Run the dialog event loop
        result = self.dlg.exec_()
        # See if OK was pressed
        if result:
            # Do something useful here - delete the line containing pass and
            # substitute with your code.
            # get working directory before assessed with pushButton
            wd = self.dlg.lineEdit.text()
            #define tiles from lineEdit efore assessed via pushButton_ext
            tiles = self.dlg.lineEdit_2.text()
            tiles = tiles.split(',')
            tiles = [t.replace(' ', '')for t in tiles]
            #get user and PW from lineEdit
            user = self.dlg.lineEdit_UN.text()
            password = self.dlg.lineEdit_PW.text()

            #get suffix for file name from lineEdit_suf
            suffix = self.dlg.lineEdit_suf.text()

            # define enndate and today from dateEdit
            start_var = self.dlg.dateEdit_start.date()
            enddate = start_var.toPyDate()
            enddate = enddate.strftime("%Y-%m-%d")

            end_var = self.dlg.dateEdit_end.date()
            today = end_var.toPyDate()
            today = today.strftime("%Y-%m-%d")

            #get the information from comboBox of data product type
            productIn = str(self.dlg.comboBox.currentText())
            if productIn == '1km 1 day':
                product = 'A1.006'
            if productIn == '1km 8 days':
                product = 'A2.006'
            #choose 'A2.006' for 8day or 'A1.006' for one day version
            #product='A1.006'
            #if checkBoxes are activated hand in parameters of satellite type
            if self.dlg.checkBox_aq.isChecked():
                MYD = True
            else:
                MYD = False
            if self.dlg.checkBox_ter.isChecked():
                MOD = True
            else:
                MOD = False
            #set True to download and select satellite type, must select one
to download anything

            if self.dlg.checkBox_del.isChecked():
                delete = True
            else:
                delete = False


##############################################################################
######

##############################################################################
######

            #https://stackoverflow.com/questions/56764046/gdal-ogr2ogr-can-
not-find-proj-db-error
            # create new variavle in windows environemnt PROJ_LIB and save
value
            #C:\Users\Jonas\.conda\envs\modis\Library\share\proj what is the
path of proj.db file;
            #then reboot your computer
            if product == 'A2.006':
                Path(wd+"/8day_mean").mkdir(parents=True, exist_ok=True)
                wd = wd+"/8day_mean"
            if product =='A1.006':
                Path(wd+"/oneday").mkdir(parents=True, exist_ok=True)
                wd = wd+"/oneday"
            #create folder for unmerged tiles
            Path(wd+"/unmergedTiles").mkdir(parents=True, exist_ok=True)
            wd = wd+"/unmergedTiles"

            #define function to download from usgs server
            def download (path, product, Tile, wd, today, enddate, user,
password):
                modisOgg = downmodis.downModis(destinationFolder=wd, pass-
word=password, user=user,
                                url='https://e4ftl01.cr.usgs.gov', tiles=Tile,
path=path,
                                product=product, today=today, enddate=enddate,
delta=0,
                                jpg=False, debug=True, timeout=None, check-
gdal=True)

                modisOgg.connect()
                modisOgg.downloadsAllDay(clean=True, allDays=False)

            #function to get array from .hdf files and spatial information
            def get_data(wd, listFile, f, int_key, kind_str):
                hdf_file = wd + '/'+listFile[f]
                scale_factor = 0.02
                # open the dataset
                hdf_ds = gdal.Open(hdf_file, gdal.GA_ReadOnly)
                subdatasets = hdf_ds.GetSubDatasets()

                subdataset_name = subdatasets[int_key][0]
                band_ds = gdal.Open(subdataset_name, gdal.GA_ReadOnly)
                band_path = os.path.join(wd, listFile[f][:-4]+ kind_str +
'.tiff')
                array = band_ds.ReadAsArray()
                array = scale_factor * array
                array = array.astype(float)
                array[array == 0] = np.nan
                return array, band_ds, band_path

            #function to create a georeferenced .tiff from array
            def to_tiff(array, band_path, band_ds):
                out_ds = gdal.GetDriverByName('GTiff').Create(band_path,
                                                              band_ds.Ras-
terXSize,
                                                              band_ds.Raster-
YSize,
                                                              1,

gdal.GDT_Float32,
                                                              ['COM-
PRESS=LZW', 'TILED=YES'])
                proj = 'PROJCS["unnamed",GEOGCS["Unknown datum based upon the
custom spheroid",DATUM["Not_specified_based_on_custom_spheroid",SPHE-
ROID["Custom spheroid",6371007.181,0]],PRIMEM["Greenwich",0],UNIT["de-
gree",0.0174532925199433,AUTHORITY["EPSG","9122"]]],PROJECTION["Sinusoi-
dal"],PARAMETER["longitude_of_center",0],PARAMETER["false_easting",0],PARAME-
TER["false_northing",0],UNIT["Meter",1],AXIS["Easting",EAST],AXIS["North-
ing",NORTH]]'
                out_ds.SetGeoTransform(band_ds.GetGeoTransform())
                out_ds.SetProjection(proj)
                out_ds.GetRasterBand(1).WriteArray(array)
                out_ds.GetRasterBand(1).SetNoDataValue(0)
                out_ds.FlushCache()
                out_ds = None  #close dataset to write to disc

            #try and delete every useless file downloaded
            def delete_down (wd):
                listdir = os.listdir(wd+ '/unmergedTiles')
                listdir_merged = os.listdir(wd + '/mergedTiles')
                listdir = [o for o in listdir if o.endswith('.hdf') or
o.endswith('.xml') or o.endswith('.txt') or o.endswith('.log') or
o.endswith('day.tiff') or o.endswith('night.tiff')] # get list of all .hdf,
.xml, .log and .txt in folder
                listdir_merged = [o for o in listdir_merged if
o.endswith('day.tiff') or o.endswith('night.tiff')]
                #try to delete all files if error raises pass
                for fd in listdir:
                    try:
                        os.remove(wd + '/unmergedTiles/' + fd)
                    except:
                        pass
                for fd in listdir_merged:
                    try:
                        os.remove(wd + '/mergedTiles/' + fd)
                    except:
                        pass
            #read information from listFile file to get names of .hdf files
            def getFileList(num, wd):
                #try reading file for every data product, if not exist just
return empty list
                try:
                    listfileMOD2 = open(wd + '/listfileMOD11A'+ num
+'.006.txt')
                    listfileMOD = listfileMOD2.read().replace("\n" ,
",").split(',')
                    listfileMOD2.close()
                    del listfileMOD[-1]
                except:
                    listfileMOD = []
                try:
                    listfileMYD2 = open(wd + '/listfile-
MYD11A'+num+'.006.txt')
                    listfileMYD = listfileMYD2.read().replace("\n" ,
",").split(',')
                    listfileMYD2.close()
                    del listfileMYD[-1]
                except:
                    listfileMYD = []
                #merge both lists
                listfileMOD = [s for s in listfileMOD if s.endswith('.hdf')]
                listfileMYD = [s for s in listfileMYD if s.endswith('.hdf')]
                listFile = listfileMOD + listfileMYD
                return listFile

##################################################################

##################################################################

##################################################################
            #depending on the input, tiles can have different datatypes,
            #coerce them to list
            if type(tiles) == tuple:
                tiles = list(tiles)
            elif type(tiles) == list:
                tiles = tiles
            elif type(tiles) == str:
                tiles = list([tiles])
            else:
                tiles = tiles


            #depending on how many tiles we want to download, download files
            for Tile in tiles:
                if MOD == True:
                    #MOD11A2 ADDDDDDDD
                    '''FOR MODIS TERRA SATALLITE'''
```

```
                        download('MOLT', 'MOD11'+product, Tile, wd, today,
enddate, user, password)
                    if MYD == True:
                        #MYD11A2 ADDDDDDDDD
                        '''FOR MODIS AQUA'''
                        download('MOLA', 'MYD11'+product, Tile, wd, today,
enddate, user, password)
                    if MYD == False and MOD == False:
                        #if no input in CheckBoxes was given, push message
                        self.iface.messageBar().pushMessage(
                        "Missing input", "Must select TERRA and/or AQUA satellite
",
                        level=Qgis.Critical, duration=3)


##############################################################
                    #depending on product type get lists with filenames
                    if product == 'A2.006':
                        listFile = getFileList('2', wd)
                    if product == 'A1.006':
                        listFile = getFileList('1', wd)


###############################################################
                    # for every file in list, get arrays for night and day and
write them to tiff
                    for f in range(len(listFile)):


##########################################################
                        array_day, band_ds, band_path = get_data(wd, listFile, f,
int_key=0, kind_str='LST_day')
                        #array_day = get_data(int_key=0, kind_str='LST_day')
                        to_tiff(array_day, band_path, band_ds)

                        array_night, band_ds, band_path = get_data(wd, listFile,
f, int_key=4, kind_str='LST_night')
                        to_tiff(array_night, band_path, band_ds)

###############################################################
                del listFile, MOD, MYD, tiles
                del password, user


###################################################################
                # merge tiles

###################################################################
                # /unmergedTiles directory back to original wd
                wd = wd[:-14]
                # Define directories for merged Tiles
                Path(wd+"/mergedTiles").mkdir(parents=True, exist_ok=True)
                out_dir = wd + '/mergedTiles/'

                # Get tif files in wd
                tile_list = glob.glob(wd+'/unmergedTiles' + '/*LST_*.tiff')
```

```
            #get sublists and zip to get unique merged- filenames
            str_start = [s.split('\\')[1][:17] for s in tile_list]
            str_end = [s.split('\\')[1].split('_')[1] for s in tile_list]
            res = np.unique([i + j for i, j in zip(str_start, str_end)])

            #https://stackoverflow.com/questions/1124810/how-can-i-find-path-
to-given-file
            #get username of this PC
            getuser = getpass.getuser()
            #get current path where plugin is installed
            py = 'MODIS.py'
            for root, dirs, files in os.walk(r'C:/Users/'+getuser+'/Ap-
pData/Roaming/QGIS/QGIS3/profiles'):
                for name in files:
                    if name == py:
                        pluginPath = os.path.abspath(os.path.join(root,
name))
            pluginPath = pluginPath.replace("\\", "/")[:-9]
            #set this path as current wd to enable gdal_merge.py to be found
            os.chdir(pluginPath)
            #merge files with same tilename
            for label in res:
                files = [wd+'/unmergedTiles/' + s.split('\\')[1] for s in
tile_list if s.endswith(label[17:]) and s.split('\\')[1][:17] == label[:17]]
                out_path = out_dir + label
                cmd = 'python ' + "gdal_merge.py" +' -co COMPRESSED=YES'+ ' -
o' + ' ' + out_path + ' ' + '-of gtiff ' + " ".join(files)
                output = subprocess.check_output(cmd)

            #Delete Loop Variables
            del cmd, output, out_path, label, files

            # Delete the other Variables
            del str_start,str_end, res, out_dir, tile_list



####################################################################
            #MEAN of oneday files: night, all and day

####################################################################
            '''define fun to make a list_stack of all files in list, get mean
            array from stack and write to tiff'''
            def mean(liste, add, suffix):
                list_stack = []
                for i in liste:
                    ds = gdal.Open(wd_mer + i)
                    bounds = ds.GetGeoTransform()
                    myarray = np.array(ds.GetRasterBand(1).ReadAsArray())
                    ds = None
                    list_stack.append(myarray)

                mean = np.nanmean(list_stack, axis=0)
                mean = mean.astype(float)
                out_ds = gdal.GetDriverByName('GTiff').Create(wd_mer+add+
enddate+'_'+today+suffix+'.tiff'.replace(' ', ''),
                                                           mean.shape[1],
                                                            mean.shape[0],
                                                            1,
                                                            gdal.GDT_Float32,
                                                            ['COMPRESS=LZW',
'TILED=YES'])
                proj = 'PROJCS["unnamed",GEOGCS["Unknown datum based upon the
custom spheroid",DATUM["Not_specified_based_on_custom_spheroid",SPHE-
ROID["Custom spheroid",6371007.181,0]],PRIMEM["Greenwich",0],UNIT["de-
gree",0.0174532925199433,AUTHORITY["EPSG","9122"]]],PROJECTION["Sinusoi-
dal"],PARAMETER["longitude_of_center",0],PARAMETER["false_easting",0],PARAME-
TER["false_northing",0],UNIT["Meter",1],AXIS["Easting",EAST],AXIS["North-
ing",NORTH]]'
                out_ds.SetGeoTransform(bounds)
                out_ds.SetProjection(proj)
                out_ds.GetRasterBand(1).WriteArray(mean)
                out_ds.FlushCache()
                out_ds = None  #close dataset to write to disc
                return list_stack, mean
            '''create .tiff to output file with percentage of existing pixel
            values which are not NAN'''
            def sum_tiff(liste, index, add, suffix):
                for i in liste:
                    i[i > 0] = 1
                summ = np.nansum(liste,0)/len(liste)*100#count existance of
values
                ds = gdal.Open(wd_mer + list_sum[index])###
                bounds = ds.GetGeoTransform()
                ds = None
                out_ds = gdal.GetDriverByName('GTiff').Cre-
ate(wd_mer+'COUNT_VALUE_%_'+add+ enddate +'_'+ today+suffix+'.tiff',#suffix
from lineEdit
                                                                summ.shape[1],
                                                                summ.shape[0],
                                                                1,

gdal.GDT_Float32,
                                                                ['COM-
PRESS=LZW', 'TILED=YES'])
                proj = 'PROJCS["unnamed",GEOGCS["Unknown datum based upon the
custom spheroid",DATUM["Not_specified_based_on_custom_spheroid",SPHE-
ROID["Custom spheroid",6371007.181,0]],PRIMEM["Greenwich",0],UNIT["de-
gree",0.0174532925199433,AUTHORITY["EPSG","9122"]]],PROJECTION["Sinusoi-
dal"],PARAMETER["longitude_of_center",0],PARAMETER["false_easting",0],PARAME-
TER["false_northing",0],UNIT["Meter",1],AXIS["Easting",EAST],AXIS["North-
ing",NORTH]]'
                out_ds.SetGeoTransform(bounds)
                out_ds.SetProjection(proj)
                out_ds.GetRasterBand(1).WriteArray(summ)
                out_ds.FlushCache()
                out_ds = None
            #all merging is only happening if downloaded data is 1km1day type
            if product == 'A1.006':
                wd_mer = wd + '/mergedTiles/'
                #make a subset of all files we want to have mean from
                list_mean = os.listdir(wd_mer)
                list_mean = [i for i in list_mean if i.endswith('night.tiff')
or i.endswith('day.tiff')]
```

```
                list_mean_night = [i for i in list_mean if
i.endswith('night.tiff')]
                list_mean_day = [i for i in list_mean if
i.endswith('day.tiff')]

                #all mean
                list_stack_all, mean_all = mean(liste = list_mean, add =
'MEANall_stacked' , suffix=suffix)
                #night mean
                list_stack_night, mean_night = mean(liste = list_mean_night,
add = 'MEANnight_', suffix=suffix)
                #day mean
                list_stack_day, mean_day = mean(liste= list_mean_day, add =
'MEANday_', suffix=suffix)

                ####calculate mean from day and night mean####similar as in
function "mean"
                list_stacky = []
                #make a stack from "mean" function returned arrays night and
day mean
                list_stacky.append(mean_night)
                list_stacky.append(mean_day)
                #mean of means (for more exact results)
                mean_all = np.nanmean(list_stacky, axis=0)
                mean_all = mean_all.astype(float)
                #write new mean to tiff
                out_ds = gdal.GetDriverByName('GTiff').Create(wd_mer+'MEAN-
dayNight(morePrecise)'+ enddate+'_'+today+suffix+'.tiff'.replace(' ', ''),
                                                          mean_all.shape[1],
                                                          mean_all.shape[0],
                                                          1,
                                                          gdal.GDT_Float32,
                                                          ['COMPRESS=LZW',
'TILED=YES'])
                proj = 'PROJCS["unnamed",GEOGCS["Unknown datum based upon the
custom spheroid",DATUM["Not_specified_based_on_custom_spheroid",SPHE-
ROID["Custom spheroid",6371007.181,0]],PRIMEM["Greenwich",0],UNIT["de-
gree",0.0174532925199433,AUTHORITY["EPSG","9122"]]],PROJECTION["Sinusoi-
dal"],PARAMETER["longitude_of_center",0],PARAMETER["false_easting",0],PARAME-
TER["false_northing",0],UNIT["Meter",1],AXIS["Easting",EAST],AXIS["North-
ing",NORTH]]'
                ds = gdal.Open(wd_mer + list_mean[0])
                bounds = ds.GetGeoTransform()
                out_ds.SetGeoTransform(bounds)
                out_ds.SetProjection(proj)
                out_ds.GetRasterBand(1).WriteArray(mean_all)
                out_ds.FlushCache()
                del ds, bounds, proj, mean_all, list_stacky
                out_ds = None

################################################################
                '''count appearance of values frequency of values which are
not
                nan per pixel in percent'''

################################################################
```

```
                #get all MEAN files for spatial information inm sum_tiff
function
                list_sum = os.listdir(wd_mer)
                list_sum = [i for i in list_sum if i[0:4] == 'MEAN' and
i.endswith(suffix+'.tiff')]

                sum_tiff(liste = list_stack_all, index = 0, add = 'all', suf-
fix=suffix)
                sum_tiff(liste = list_stack_day, index = 1, add = 'day', suf-
fix=suffix)
                sum_tiff(liste = list_stack_night, index = 2, add = 'night',
suffix=suffix)

            #delete everything in 1km1day what litters your memory
            #delete all unneccessary files

            if delete == True and product == 'A1.006':
                delete_down(wd)

            if delete == True and product == 'A2.006':
                listdir_8 = os.listdir(wd+ '/unmergedTiles')
                listdir_8 = [o for o in listdir_8 if o.endswith('.hdf') or
o.endswith('.xml') or o.endswith('.txt') or o.endswith('.log')]
                # get list of all .hdf, .xml, .log and .txt in folder
                #try to delete all files if error raises pass
                for fd in listdir_8:
                    try:
                        os.remove(wd + '/unmergedTiles/' + fd)
                    except:
                        pass

            del product
#
==============================================================================
#             wd_to_tif = wd + '/mergedTiles'
#             wd_to_tif = [x for x in os.listdir(wd_to_tif) if x.starts-
with('MEAN') or x.startswith('COUNT')]
#
#
==============================================================================
            # addLayer doesn't work because of unknown error
#
==============================================================================
#             for layer in wd_to_tif:
#                 rlayer = QgsRasterLayer(layer, layer[-52:].split('/')[1])
#                 if not rlayer.isValid():
#                     print("Layer"+layer[-52:].split('/')[1]+ "failed to
load!")
#                 self.iface.addRasterLayer(layer, layer[-52:].split('/')[1])
#
==============================================================================

            self.iface.messageBar().pushMessage(
            "Success", "Output files written at " + wd,
            level=Qgis.Success, duration=3)
            del wd
```

Bibliography

About pyModis — pyModis 2.1.0 documentation [WWW Document], n.d. URL http://www.pymodis.org/info.html (accessed 3.2.21).

Arnfield, A.J., 2003. Two decades of urban climate research: a review of turbulence, exchanges of energy and water, and the urban heat island. Int. J. Climatol. 23, 1–26. https://doi.org/10.1002/joc.859

Berk, A., 1987. MODTRAN: A MODERATE RESOLUTION MODEL FOR LOWTRAN. MODTRAN: A Moderate Resolution Model for LOWTRAN 40.

Berk, A., Bernstein, L.S., Robertson, D.C., 1987. MODTRAN: A Moderate Resolution Model for LOWTRAN. SPECTRAL SCIENCES INC BURLINGTON MA.

Caselles, V., Sobrino, JoséA., 1989. Determination of frosts in orange groves from NOAA-9 AVHRR data. Remote Sens. Environ. 29, 135–146. https://doi.org/10.1016/0034-4257(89)90022-9

Cattle, H., Crossley, J., Drewry, D.J., Wadhams, P., Dowdeswell, J.A., Schofield, A.N., 1995. Modelling Arctic climate change. Philos. Trans. R. Soc. Lond. Ser. Phys. Eng. Sci. 352, 201–213. https://doi.org/10.1098/rsta.1995.0064

EarthExplorer [WWW Document], n.d. URL https://earthexplorer.usgs.gov/.

Feng Gao, Masek, J., Schwaller, M., Hall, F., 2006. On the blending of the Landsat and MODIS surface reflectance: predicting daily Landsat surface reflectance. IEEE Trans. Geosci. Remote Sens. 44, 2207–2218. https://doi.org/10.1109/TGRS.2006.872081

Hegde, A., 2019. aishwaryahegde29/modis_nitk https://github.com/aishwaryahegde29/modis_nitk (accessed 3.23.21).

Heipke, C., Freeden, W., Rummel, R. (Eds.), 2017. Photogrammetrie und Fernerkundung, 1. Auflage. ed, Handbuch der Geodäsie. Springer Spektrum, Berlin.

Humes, K.S., Kustas, W.P., Moran, M.S., Nichols, W.D., Weltz, M.A., 1994. Variability of emissivity and surface temperature over a sparsely vegetated surface. Water Resour. Res. 30, 1299–1310. https://doi.org/10.1029/93WR03065

Isaya Ndossi, M., Avdan, U., 2016. Application of Open Source Coding Technologies in the Production of Land Surface Temperature (LST) Maps from Landsat: A PyQGIS Plugin. Remote Sens. 8, 413. https://doi.org/10.3390/rs8050413

Jackson, R.D., Reginato, R.J., Idso, S.B., 1977. Wheat canopy temperature: A practical tool for evaluating water requirements. Water Resour. Res. 13, 651–656. https://doi.org/10.1029/WR013i003p00651

Jiménez-Muñoz, J.C., Sobrino, J.A., 2003. A generalized single-channel method for retrieving land surface temperature from remote sensing data. J. Geophys. Res. Atmospheres 108, 2003JD003480. https://doi.org/10.1029/2003JD003480

Jin, M., Shepherd, J.M., Peters-Lidard, C., 2007. Development of a parameterization for simulating the urban temperature hazard using satellite observations in climate model. Nat. Hazards 43, 257–271. https://doi.org/10.1007/s11069-007-9117-2

Kuenzer, C., Dech, S. (Eds.), 2013. Thermal Infrared Remote Sensing, Remote Sensing and Digital Image Processing. Springer Netherlands, Dordrecht. https://doi.org/10.1007/978-94-007-6639-6

Laaidi, K., Zeghnoun, A., Dousset, B., Bretin, P., Vandentorren, S., Giraudet, E., Beaudeau, P., 2012. The Impact of Heat Islands on Mortality in Paris during the August 2003 Heat Wave. Environ. Health Perspect. 120, 254–259. https://doi.org/10.1289/ehp.1103532

Liu, L., Zhang, Y., 2011. Urban Heat Island Analysis Using the Landsat TM Data and ASTER Data: A Case Study in Hong Kong. Remote Sens. 3, 1535–1552. https://doi.org/10.3390/rs3071535

Matson, M., Mcclain, E.P., McGinnis, D.F., Pritchard, J.A., 1978. Satellite Detection of Urban Heat Islands. Mon. Weather Rev. 106, 1725–1734. https://doi.org/10.1175/1520-0493(1978)106<1725:SDOUHI>2.0.CO;2

MODIS Land Team Home Page [WWW Document], 2021. URL https://modis-land.gsfc.nasa.gov/MODLAND_grid.html (accessed 3.6.21).

MODIS Web [WWW Document], 2021. URL https://modis.gsfc.nasa.gov/about/index.php (accessed 4.25.21a).

MODIS Web [WWW Document], 2020. URL https://modis.gsfc.nasa.gov/data/dataprod/mod11.php (accessed 3.2.21b).Petropoulos, G., Carlson, T.N., Wooster, M.J., Islam, S., 2009. A review of Ts/VI remote sensing based methods for the retrieval of land surface energy fluxes and soil surface moisture. Prog. Phys. Geogr. Earth Environ. 33, 224–250. https://doi.org/10.1177/0309133309338997

Rahmahalim, M., 2020. oxwazz/Plugin-QGIS-SPT https://github.com/OXWAZZ/Plugin-QGIS-SPT/ (accessed 3.23.21).

Rozenstein, O., Qin, Z., Derimian, Y., Karnieli, A., 2014. Derivation of Land Surface Temperature for Landsat-8 TIRS Using a Split Window Algorithm. Sensors 14, 5768–5780. https://doi.org/10.3390/s140405768

Salisbury, J.W., D'Aria, D.M., 1992. Emissivity of terrestrial materials in the 8–14 μm atmospheric window. Remote Sens. Environ. 42, 83–106. https://doi.org/10.1016/0034-4257(92)90092-X

Senay, G., Budde, M., Verdin, J., Melesse, A., 2007. A Coupled Remote Sensing and Simplified Surface Energy Balance Approach to Estimate Actual Evapotranspiration from Irrigated Fields. Sensors 7, 979–1000. https://doi.org/10.3390/s7060979

SinusoidalMODIS – Marine Geospatial Ecology Tools [WWW Document], n.d. URL https://code.env.duke.edu/projects/mget/wiki/SinusoidalMODIS (accessed 3.2.21).

Sobrino, J.A., Zhao-Liang Li, Stoll, M.P., 1993. Impact of the atmospheric transmittanceand total water vapor content in the algorithms for estimating satellite sea surface temperatures. IEEE Trans. Geosci. Remote Sens. 31, 946–952. https://doi.org/10.1109/36.263765

Wan, Z., 2014. New refinements and validation of the collection-6 MODIS land-surface temperature/emissivity product. Remote Sens. Environ. 140, 36–45. https://doi.org/10.1016/j.rse.2013.08.027

Wan, Z., 2013. MODIS Land Surface Temperature Products 33 https://lpdaac.usgs.gov/documents/118/MOD11_User_Guide_V6.pdf (accessed last: 3-23-21).

Wan, Z., 1999. Contract Number: NAS5-31370 77 https://modis.gsfc.nasa.gov/data/atbd/atbd_mod11.pdf (accessed last: 3-23-21).

Zhang, J., Wang, Y., Li, Y., 2006. A C++ program for retrieving land surface temperature from the data of Landsat TM/ETM+ band6. Comput. Geosci. 32, 1796–1805. https://doi.org/10.1016/j.cageo.2006.05.001

Zhengming Wan, Dozier, J., 1996. A generalized split-window algorithm for retrieving land-surface temperature from space. IEEE Trans. Geosci. Remote Sens. 34, 892–905. https://doi.org/10.1109/36.508406

Zhengming Wan, Zhao-Liang Li, 1997. A physics-based algorithm for retrieving land-surface emissivity and temperature from EOS/MODIS data. IEEE Trans. Geosci. Remote Sens. 35, 980–996. https://doi.org/10.1109/36.602541

List of Figures

List of tables