The Journal of Open Source Software

ICESat-2 Altimeter Data using R

Lampros Sp. Mouselimis[1]

1 Monopteryx, Rahouli Paramythias, Thesprotia, Greece

DOI:

Software

- Review
- Repository
- Archive

Submitted:
Published:

Summary

Lidar technology (light detection and ranging) is integrated into autonomous cars, unmanned aerial vehicles (UAV), airplanes, and satellites and as of October 2021, it represents a growing market. Many satellite missions utilize lidar to observe the surface of the earth and one of these is ICESat-2, which was launched on 15th September 2018 with the primary goal to measure changes in glaciers and ice sheets (Smith et al., 2019). ICESat-2 (the successor of ICESat) consists of 6 beams (3 pairs), where each pair is separated by 3 kilometers and each beam in the pair is also separated by 90 meters as the following image shows (Smith et al., 2019),

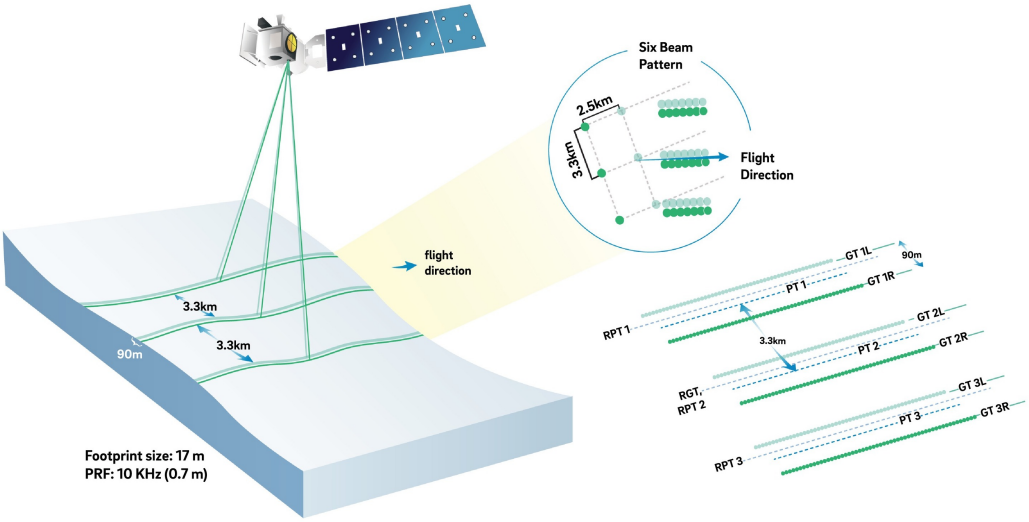

Figure 1: the ICESat-2 ATLAS six-beam pattern

Besides the Distributed Active Archive Center (DAAC) at NSIDC another source to retrieve ICESat-2 data is the OpenAltimetry platform, which allows users to discover, access, and visualize data from NASA's ICESat and ICESat-2 missions (Khalsa et al., 2020).

The IceSat2R package (Mouselimis, 2022) includes functionality that

- creates a connection to the OpenAltimetry Web API
- allows users to interactively create a global degree-grid, which is required for the OpenAltimetry queries
- makes feasible the download of the required Reference Ground Tracks (RGTs) from the National Snow & Ice Data Center (NSIDC)
- includes three vignettes that explain in detail how users can analyze ICESat-2 data and combine it with other satellite and in-situ sources.

Statement of need

Online access to data and analysis tools via easy-to-use interfaces can significantly increase data usage across a wide range of users (Khalsa et al., 2020). In the same way, authors of programming packages should incorporate the required functionality so that the biggest possible number of users can access and take advantage of the codebase. An important aspect of the IceSat2R package is that it includes the code, documentation, and examples so that users can retrieve, process, and analyze data based on specific workflows. For instance,

- A user can select an *area of interest* (AOI) either programmatically or interactively
- If the *Reference Ground Track* (RGT) is not known, the user has the option to utilize either
  - one of the **'overall_mission_orbits()'** or **'time_specific_orbits()'** to compute the RGT(s) for a pre-specified global area or for a time period, or
  - one of the **'vsi_nominal_orbits_wkt()'** or **'vsi_time_specific_orbits_wkt()'** to compute the RGT(s) for a specific AOI
- Once the RGT is computed it can be verified with the **'getTracks()'** function of the OpenAltimetry Web API
- Finally the user can utilize one of the **'get_atlas_data()'** or **'get_level3a_data()'** functions to retrieve the data for specific product(s), Date(s) and Beam(s)

This work-flow is illustrated also in the next diagram,

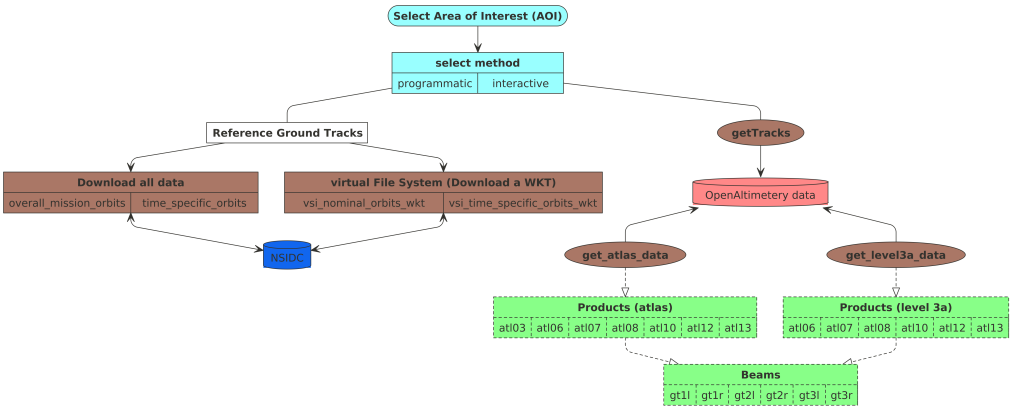

Figure 2: IceSat2R Workflow Diagram

Example use case

The example code of this section explains how an R user can utilize the IceSat2R package to,

- extract the *Reference Ground Tracks (RGT)* of an AOI (*Himalayas mountain range*) using the *GDAL Virtual File System*
- retrieve data from the *OpenAltimetry API* for a pre-specified bounding box (as described in the previous work-flow)
- report the number of downloaded and available observations for each specified ICESat-2 product

The following map shows the bounding box areas (small on the left and big on the right) that will be used in the code.

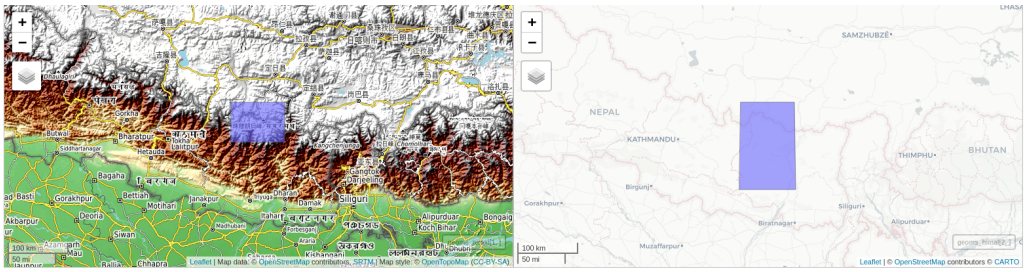

Figure 3: Himalayas AOI

```
# required R packages

pkgs = c('IceSat2R', 'magrittr', 'sf', 'mapview', 'leaflet', 'knitr', 'glue',
         'data.table', 'CopernicusDEM', 'terra', 'geodist', 'ggplot2')
load_pkgs = lapply(pkgs, require, character.only = TRUE)

# load the AOI

himl_pth = system.file('data_files', 'vignette_data', 'himalayas.RDS',
                       package = "IceSat2R")

geoms_himal = readRDS(himl_pth)

sf_wkt = sf::st_geometry(subset(geoms_himal, area_size == 'big'))
centr_wkt = sf::st_coordinates(sf::st_centroid(sf_wkt))
dat_wkt = sf::st_as_text(sf_wkt)

# iterate over all 8 available repeats for the Eastern Hemisphere

lst_out = list()

for (iter in 1:8) {
  dat_iter = IceSat2R::vsi_nominal_orbits_wkt(orbit_area = 'eastern_hemisphere',
                                              track = 'GT7',
                                              rgt_repeat = iter,
                                              wkt_filter = dat_wkt,
                                              download_method = 'curl',
                                              download_zip = FALSE,
                                              verbose = TRUE)
  lst_out[[iter]] = dat_iter
}

# Extract the unique Reference Ground Tracks (RGTs)

lst_out = unlist(lst_out, recursive = F)
unq_rgts = as.vector(unique(unlist(lapply(lst_out, function(x) x$RGT))))
unq_rgts
# [1] "96"   "157"  "363"  "538"  "599"  "805"  "866"  "1041" "1308" "1247"
```

In this use case, we are interested in ICESat-2 data for a specific time period (from *'2020-01-01'* to *'2021-01-01'* - 1-year's data). We'll make use of the *Ice-*

Sat2R::vsi_time_specific_orbits_wkt() function which queries all *15 ICESat-2 RGTs cycles* (as of March 2022) to come to the RGTs intersection for the specified 1-year time interval,

```
date_start = '2020-01-01'
date_end = '2021-01-01'

orb_cyc_multi = IceSat2R::vsi_time_specific_orbits_wkt(date_from = date_start,
                                                       date_to = date_end,
                                                       RGTs = unq_rgts,
                                                       wkt_filter = dat_wkt,
                                                       verbose = TRUE)
```

The next map shows the 18 different Date-Time matches for our defined 1-year time period based on the output of the *'IceSat2R::vsi_time_specific_orbits_wkt()'* function,

```
orbit_cy = mapview::mapview(orb_cyc_multi, legend = F)
AOI_wkt = mapview::mapview(sf_wkt, legend = F)

lft = orbit_cy + AOI_wkt

lft@map %>% leaflet::setView(lng = centr_wkt[, 'X'],
                             lat = centr_wkt[, 'Y'],
                             zoom = 7)
```

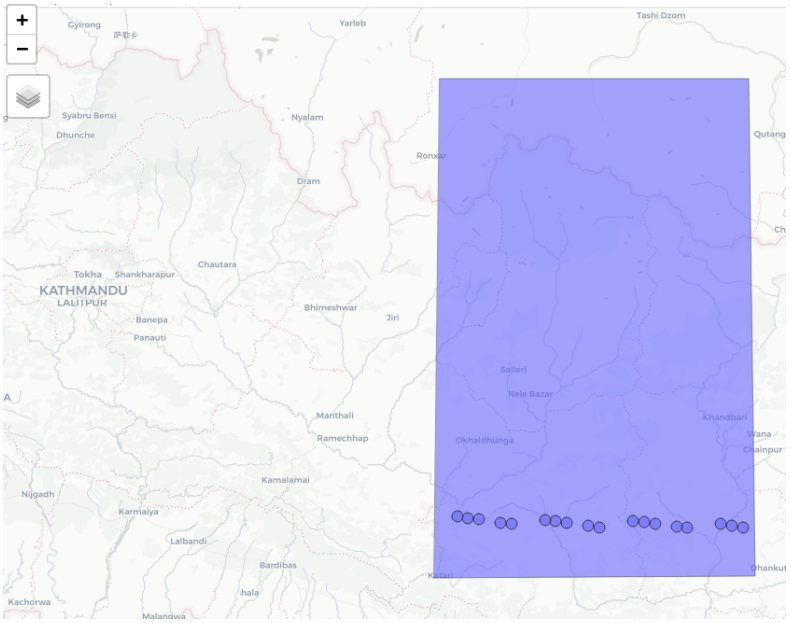

Figure 4: Intersection of the AOI and the ICESat-2 Orbits

The output of *'vsi_time_specific_orbits_wkt()'* can be verified with the *OpenAltimetry's 'getTracks()'* function,

```
bbx_aoi = sf::st_bbox(obj = sf_wkt)
rows_match = rep(FALSE, nrow(orb_cyc_multi))

for (row in 1:nrow(orb_cyc_multi)) {

  dat_item = orb_cyc_multi[row, , drop = F]
  Date = as.Date(dat_item$Date_time)
```

```
  op_tra = IceSat2R::getTracks(minx = as.numeric(bbx_aoi['xmin']),
                               miny = as.numeric(bbx_aoi['ymin']),
                               maxx = as.numeric(bbx_aoi['xmax']),
                               maxy = as.numeric(bbx_aoi['ymax']),
                               date = as.character(Date),
                               outputFormat = 'csv',
                               download_method = 'curl',
                               verbose = FALSE)

  date_obj = dat_item$Date_time
  tim_rgt = glue::glue("Date: {date_obj} Time specific RGT: '{dat_item$RGT}'")

  if (nrow(op_tra) > 0) {
    iter_op_trac = paste(op_tra$track, collapse = ', ')
    cat(glue::glue("{tim_rgt}  OpenAltimetry: '{iter_op_trac}'"), '\n')

    if (length(op_tra$track) == 1) {
      if (op_tra$track == dat_item$RGT) {
        rows_match[row] = TRUE
      }
    }
  }
  else {
    cat(glue::glue("{tim_rgt} without an OpenAltimetry match!"), '\n')
  }
}
```

We keep the rows that intersect with the output of the *OpenAltimetry getTracks()* function,

```
inters_opnalt_rgt = orb_cyc_multi[rows_match, , drop = F]
```

We also restrict our initial AOI to a smaller area in the *Himalayas mountain range*,

```
sf_wkt_init = sf::st_geometry(subset(geoms_himal, area_size == 'small'))
bbx_aoi_init = sf::st_bbox(obj = sf_wkt_init)
```

A potential use case would be to visualize the *Ice*, *Land* and *Canopy* height differences, and the right function for this purpose would be the *IceSat2R::get_level3a_data()* function that takes a *time interval* as input. The corresponding ICESat-2 and OpenAltimetry API products are the **'atl06'** and **'atl08'**,

```
RGTs = sort(unique(inters_opnalt_rgt$RGT))
Products = c('atl06', 'atl08')
```

We'll iterate over the mentioned Products and available tracks to retrieve the altimeter data,

```
dat_out = logs_out = list()

for (prod_i in Products) {
  for (track_i in RGTs) {

    name_iter = glue::glue("{track_i}_{prod_i}")
    cat(glue::glue("RGT: '{track_i}'  Product: '{prod_i}'"), '\n')

    minx = as.numeric(bbx_aoi_init['xmin'])
    miny = as.numeric(bbx_aoi_init['ymin'])
```

```
    maxx = as.numeric(bbx_aoi_init['xmax'])
    maxy = as.numeric(bbx_aoi_init['ymax'])

    iter_dat = IceSat2R::get_level3a_data(minx = minx,
                                          miny = miny,
                                          maxx = maxx,
                                          maxy = maxy,
                                          startDate = date_start,
                                          endDate = date_end,
                                          trackId = track_i,
                                          beamName = NULL,
                                          product = prod_i,
                                          client = 'portal',
                                          outputFormat = 'csv',
                                          verbose = FALSE)

    iter_logs = list(RGT = track_i,
                     Product = prod_i,
                     N_rows = nrow(iter_dat))

    logs_out[[name_iter]] = data.table::setDT(iter_logs)
    dat_out[[name_iter]] = iter_dat
  }
}
```

Finally, the logs in the following table show the RGTs and the retrieved number of rows for the specified ICESat-2 products.

```
# output logs

dtbl_logs = data.table::rbindlist(logs_out)
dtbl_logs = subset(dtbl_logs, N_rows > 0)
dtbl_logs = dtbl_logs[order(dtbl_logs$N_rows, decreasing = T), ]
dtbl_logs

#      RGT Product N_rows
#    <int>  <char>  <int>
# 1:   599   atl06  56531
# 2:    96   atl06  51874
# 3:  1041   atl06  49208
# 4:   599   atl08  10250
# 5:  1041   atl08   9205
# 6:    96   atl08   6006
# 7:   538   atl06   1535
# 8:   538   atl08    201
```

Acknowledgements

The development of the IceSat2R package was supported by the R Consortium (grant code 21-ISC-2-2).

References

Khalsa, S. J. S., Borsa, A., Nandigam, V., Phan, M., Lin, K., Crosby, C., Fricker, H., et al. (2020). OpenAltimetry - rapid analysis and visualization of Spaceborne altimeter data. *Earth Science Informatics*. doi:10.1007/s12145-020-00520-2

Mouselimis, L. (2022). *IceSat2R: ICESat-2 altimeter data using r*. Retrieved from https://CRAN.R-project.org/package=IceSat2R

Smith, B., Fricker, H. A., Holschuh, N., Gardner, A. S., Adusumilli, S., Brunt, K. M., Csatho, B., et al. (2019). Land ice height-retrieval algorithm for NASA's ICESat-2 photon-counting laser altimeter. *Remote Sensing of Environment*, *233*, 111352. doi:10.1016/j.rse.2019.111352