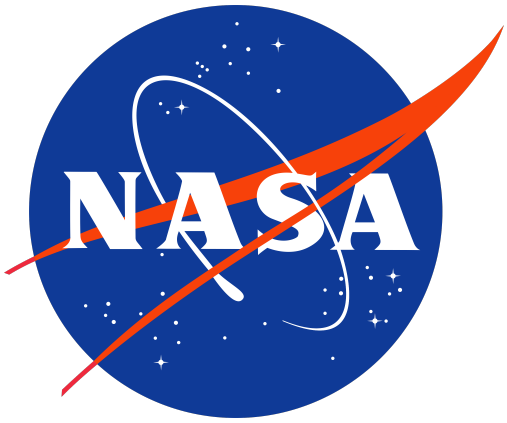

Software Utility Suite for the Sentinel-6 Michael Freilich and Sentinel-6B Supplemental Calibration Systems

Final Report

August 12th, 2024

by

Pablo Cesar Bedolla Ortiz
pablo.cesar.bedolla.ortiz@jpl.nasa.gov

JPL Instrument Operations Engineering Group (398D)

Dominican University
Illinois Institute of Technology

Under the supervision of

Dr. Daniel Wenkert
Ramona Tung
Devin Johnson

NASA Jet Propulsion Laboratory
California Institute of Technology

Contents

1 Abstract

The Sentinel-6 series of Earth observation satellites will be used to monitor sea level changes, track ocean currents, enhance climate models, and provide long-term data, helping assess the health of the planet's oceans and contribute to climate and environmental research. Located on the spacecraft, the Advanced Microwave Radiometer-Climate Quality (AMR-C) instrument, which handles tropospheric corrections, includes the Supplemental Calibration System for radiometric calibrations. At five-day intervals, two known calibration target values are used to perform internal system corrections and ensure their functionality. Telemetry collected during pre-launch test phases and transmitted during the mission includes data on defined service intervals performed by the calibration subsystem, which should be monitored for anomaly detection and health assessment, keeping track of subsystem performance decline over time. The existing software tool for SCS parameter trending contains significant bugs and errors, lacks modularity for future mission adaptability, and is not suited for effective team handover for future missions. A new and improved framework is presented, the Supplemental Calibration System—Know, Navigate, Integrate, and Track (SCS-KNIT) kit, granting extended functionality over parameter trending with surpassed speed and performance benchmarks for functionality consistently used by the operations team, compared to other software within the branch. With the development of SCS-KNIT, AMR-C instrument continuity can maintain effective and efficient long-term health monitoring with fast and reliable performance.

2 Overview

Launched on the twenty-first of November in 2020 from Vandenburg Air Force Base, the Sentinel-6 Michael Freilich deployed aiming to provide information regarding Earth's ocean circulation, climate change, and sea-level fluctuations (1). Two identical satellites, Sentinel-6 Michael Freilich and Sentinel-6B, scheduled for launch in 2025, will operate together as a unified pair for the Sentinel-6 series of Earth observation systems (2). On both satellites, the Advanced Microwave Radiometer-Climate Quality (AMR-C) instrument is integrated within the altimeter payload to perform tropospheric path delay corrections (4; 5). An internal subsystem, the Supplemental Calibration System (SCS) is embedded to improve the measurement quality for more accurate and faster radiometer calibrations using two targets of known values (1). Each calibration, conducted at five-day intervals, the system calibrates internal radiometer calibration sources using the cold space and a warm target. Throughout various testing phases and during flight, telemetry data is collected and tracked monitoring the motions performed by the calibration system. Each motion type performed by the calibration system is defined on a specific interval. These motions should be tracked and monitored to analyze the health of the subsystem, observing the overall degradation of the instrument over time. Comparison between pre and post-flight data ensures that the system is performing to its defined benchmarks and mission continuity is sustained. Within this subsystem monitoring loop, the SCS subsetter script is deployed. The compact toolkit provides adequate-working functionality to extract SCS motions providing a master excel file with derived statistics from the telemetry parameters alongside for trending. However, this software kit is limited in its capabilities nonetheless error prone to version changes. The framework is hard-coded and unadaptable to mission transitions requiring similar applications, rendering it non-modular and non-reusable. A new software framework, known as the Supplemental Calibration System—Know, Navigate, Integrate, Track (SCS-KNIT) framework, is introduced to the Sentinel-6 series operations teams for evaluating the SCS motions and their derived statistics. The framework encompasses a multitude of console and graphical tooling capabilities providing operations members with modular building blocks to assess, analyze, and compare pre and post-flight SCS motions. The extended functionality on parameter trending alongside with overshadowed speed and performance benchmarks for logic will enhance operations efficiency as it will observations on the health of the system. The AMR-C instrument can ensure mission continuity and maintain an effective and prolonged performance throughout its mission lifespan.

3 Statement of Objective and Scope

The intended objective of the SCS-KNIT framework is to provide instrument operations members for the Sentinel-6 series with a toolkit that can track and monitor calibration motions. Though this merely represents

the surface of the core purpose of the software kit, monitoring the health and status of the subsystem is critical. The KNIT framework delivers a plethora of tools to not only perform this analysis, but to also graphical observe it, package it for delivery to other members, and standard tasks like searching directories, processing telemetry, and others.

3.1 Issues with Subsetter

A multitude of issues emerge with the original SCS motion tracker when considering it from a long-term perspective and within the broader context, particularly with expanded team delivery. The SCS subsetter script is an internal script designed to query SCS datasets, process the data, and identify SCS movements to produce statistical outputs. These outputs are then delivered to operations members to make spacecraft maneuver decisions based on their observations. However, there are a few issues associated with this long-standing framework.

3.1.1 Insufficient Runtime Speeds

In a benchmark comparison, the logic of simple file retrieval utilities outperforms the original script in speed by 195%. The original script employs a repetitive use of recursive file retrieval methods in an iterative manner, such as *rglob*, which can lead to excessive deep recursion. Consequently, stack overflows are a risk due to running out of RAM when handling large files and reaching the maximum search depth.

3.1.2 Insufficient Data Processing

When dealing with thousands of files, each containing thousands of rows of data, data processing becomes essential. Well-defined data structures ensure that the internal scripts used by the toolkit function as intended, facilitating easier unit testing. Nonetheless, retrieving files and using them immediately bypasses a crucial step: verifying that the format is usable and that all entries in a data file are valid. Additional found issues in the data cleaning process of subsetter indicate that groups of retrieved files contain vast duplicate data, null values, and incompatible types. Although the script uses this data and manages to work past these errors, the time and space usage of the script working through these files adds extra strain to the procedure. A clear example of this issue is the implementation of a skip master file. This file is intended to list filenames to exclude when running the tool. However, a single-line error causes every file read to be continuously added to the skip file during each iteration of the tool. As a result, the skip file grows to over 34,000 entries, compared to the actual number of skip and non-skip files, which is about 1,700. This excessive accumulation strains the program's runtime performance and can lead to additional RAM memory issues.

3.1.3 Absence of Batching

A massive issue with the SCS subsetter framework is the lack of data batch processing. Part of the tool's procedure involves collecting files with a specific keyword and extension from an SMB network drive share. All these files are then merged together to create a master file and prepared for future use. Upon completion, the master file contains over 15 million entries, placing an extreme strain on a computer with minimal system resources. This can lead to buffer overflow or crash issues, emphasizing the need for batch processing to both improve runtime speeds and ensure the program runs without sudden crashes. The absence of parallelization for large and intensive tasks will result in slow runtime speeds.

3.1.4 Readability and Modularity

Modular software is critical when aimed to be used in large teams. Nonetheless, distribution of the software to other teams requires modular functionality to accustom to that teams needs. The SCS subsetter framework is a monolithic script with tightly coupled components and minimal abstraction. Aside from its complexity and difficult to maintain structure, it is prone to many errors due to its difficulty to refactor.

3.1.5 Concealed Irregularities

One of the key features of the SCS-KNIT toolkit is its motion extraction algorithm. This procedure searches for successive pairs of matching state statuses within the calibration system and uses a divide-and-conquer approach to ensure that every nested status is correctly ordered. This method indirectly confirms that all state statuses are identified, thereby enabling accurate motion extraction. However, prior to taking a divide-and-conquer approach, merely verifying that state statuses exists within a motion window is not sufficient to verify the motion is legal (Figure 1 & 2).

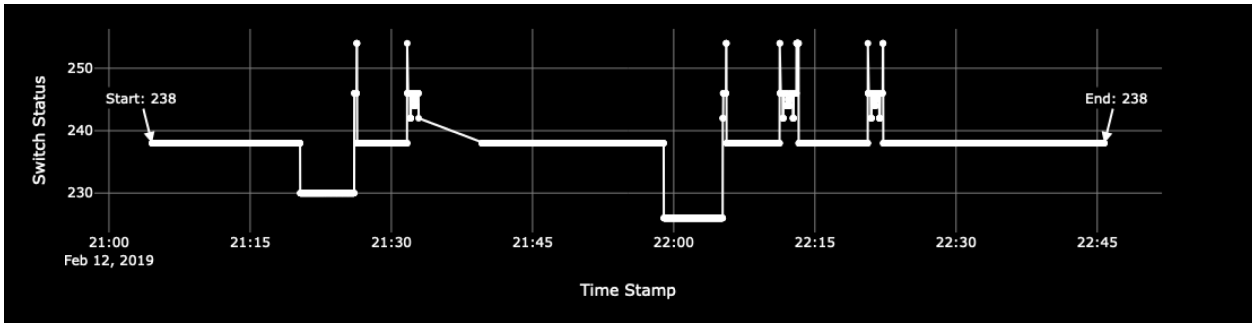

Figure 1: Pre-complete motion extracted using the SCS-KNIT algorithm, with two nested motions shown. All state statuses are present within this window.

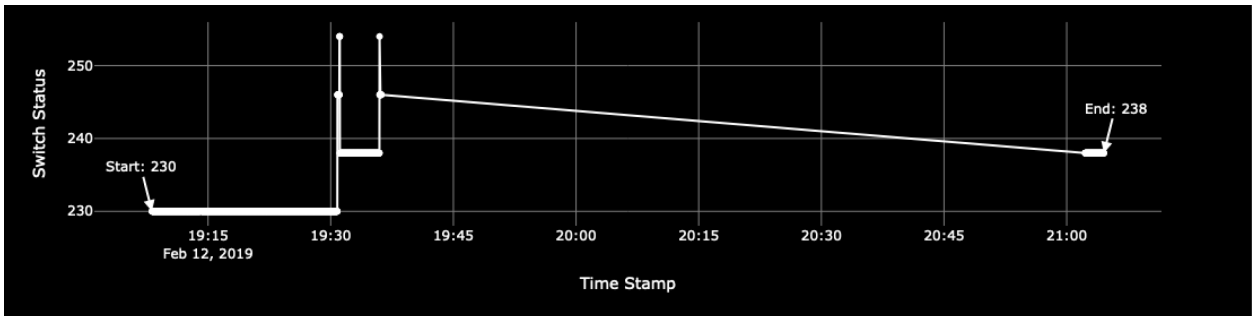

Figure 2: Pre-complete motion extracted using the SCS-KNIT algorithm. All state statuses are present within this window.

4 Know, Navigate, Integrate, Track (KNIT)

4.1 Modularity

The SCS-KNIT framework offers extensive modularity enabling developers to select their own logic blocks, classes, and functions to extract motions at their desire. Instrument operations developers are not constrained to a single script for extracting motions or utilizing logic within the framework. The toolkit offers extensive functionality beyond motion extraction, including graphical plotting tools, data class abstraction, file retrieval, motion extraction with custom statuses, enum constants for future applications, and more.

4.2 Algorithms

The highlight of the SCS-KNIT toolkit is its three master algorithms for finding specific state motions. These three algorithms can identify calibration motions that meet all three target types: full motions, transitions to one target type (Forward half), and transitions to the other target type (Backward half).

4.2.1 Full Motion Extraction

A full extraction motion is defined as a full calibration state transition touching all position switches at Science, Space View Mirror (SVM), and Warm Calibration Target (WCT). The full calibration command sequences for Supplemental Calibration System (SCS) A or redundant B (spare) follow the exact same sequence but separate status values. These motions follow a sequential order of status transitions, making them easily identifiable. The challenge in identifying sequential ordered pairs lies in the fact that the order of statuses can adhere to all legal transitions until it deviates (Figure 3). The process starts by identifying the outermost status and searching for it within rows of data entries. Once found, it records data until the status appears again, defining that segment as a motion. To fully categorize it as a motion, the algorithm continues to search for all nested pairs of successive statuses, progressively narrowing down the search in a manner akin to a Matryoshka doll. This approach involves breaking down complex problems into nested, simpler sub-problems, much like the nested structure of Matryoshka dolls, with each sub-problem addressed individually (Figure 4 & 5).

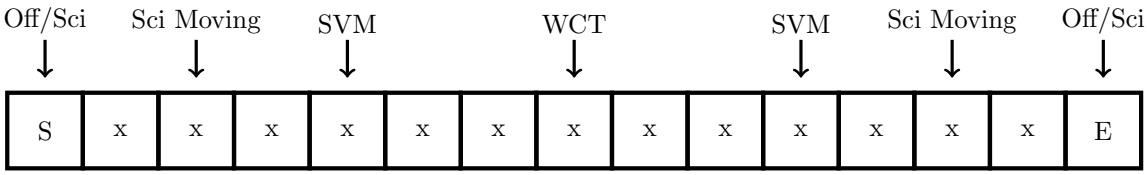

Figure 3: Illustration of the sequence of states in the system.

Figure 4: Part A: Sequence of states in the system outermost.

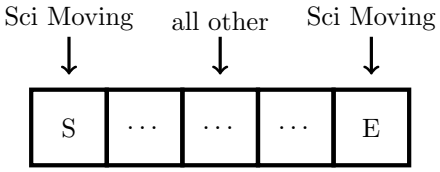

Figure 5: Part B: Sequence of states in the system innermost.

4.2.2 Constants

The constants module provides operations members with argument adjustment files to create and pre-define arguments used by the tool. If a member is interested in changing the start and end state statuses of a motion, an integer change is only required rather than a full-scale script refactor. The constants modules is not confined to only the class constants already defined such as packet types, switch statuses (decimal and hexadecimal), search column names, indicators and more. Operations members are encouraged to further develop the constants module and abstract any hard-coded logic within the program. This will help the module serve not only as documentation but also as human-readable code with decoupled logic. The constants module includes additional sub-modules including *Paths*, *Helpers*, *Colors*, and *AMR-C Statuses*. These sub-modules support development by providing constants for graphical plotting, paths for setting folders and files for input, output, and search procedures, as well as utility functions for managing constants.

The graphical plotting module will allow operations member to abstract from being tied down to using excel spreadsheets only for observation and will enable cross-functionality with web frameworks to run additive logic on-top of the modules functionality.

4.2.3 Models

The *Models* module encompasses all the necessary functionality for extracting and storing state motions in an SCS calibration. Each defined state for a series of SCS calibration transitions has its own dataclass, which stores a variety of data entries including start and end switches, times, current statistics, durations, and more. Each state is divided into its own class holding information for that specific state in a given window. To simplify the development process, developers use functions associated with the class to define start and end states, as for additional fields, eliminating the need to define each field individually. Each class also includes additional logic to print the data set and export it to structured JSON format. Lastly, the major function of every class is the algorithm to find that specific state. Although the algorithm itself does not exist as a standalone entity, its underlying logic is applied in every state class and is also utilized in the master extraction of full and half motions.

4.2.4 Utilities

The *Utilities* module contains the most critical functionality. In here, a sub-module can be found which contains logic to find, extract, and generate state motions and data. The *generator* sub-module includes all logic to generate housekeeping and motor current DataFrame's which are required to find motions. In this module, you can also find the master algorithms to extract motions. The *helper* sub-module encompasses logic for SBM network connection functions, regex functions, increasing sub-sequence algorithms, and various other utility functions. It is designed to automate processes and reduce redundant code. The *searching* module contains similar functionality specific to file search and I/O logic. Lastly, the *cleaning* and *plots* modules include functionality for preprocessing data and graphical plotting.

4.3 Efficiency

In addition to incorporating new functionality and logic, improvements in speed and efficiency are also prioritized. The original SCS subsetter code lacked data batch processing, parallelization, and intuitive iterative approaches. Nonetheless, defined logic within the framework often deemed code unreachable or redundant, creating additive strain to the program to both memory and speed. A common issue with the subsetter is the use of raw recursion logic. Raw recursion can be costly because each recursive call requires a new stack frame. While parallelizing recursion is challenging due to its reliance on previous results, this is not a concern with file searches, where parallelization can be effectively utilized. Using a more robust and faster library for file searching—such as *find*, which is extensively utilized in this toolkit—can significantly enhance performance (7). Written in the faster Rust language, *find* offers superior speed compared to the Unix *fd* tool. It enables efficient multi-search operations with parallelization and handles repetitive checks in preprocessing effectively.

Comparing a single-use case of raw recursion versus parallelized recursive directory traversal, *find* outshines (Table 1). The two benchmarks are compared with internet speeds outside of lab as the search run time varies based on the network connectivity.

	Raw Recursive	Parallelized
Time	2h 4m 55s	1m 40s

Table 1: Comparison of Recursive and Parallelized Methods

Iterative methods have also surpassed the original logic of the subsetter in terms of speed. By switching to iterating over large rows of data entries in a specific column, enumerating it, and using an index for O(1) searches, performance is significantly improved. Among the methods df.iterrows(), df.itertuples(), and `df[column]`, the direct column call method is nearly 200% (198.91%) faster and about 129% (128.85%) faster than df.itertuples() (figure 6).

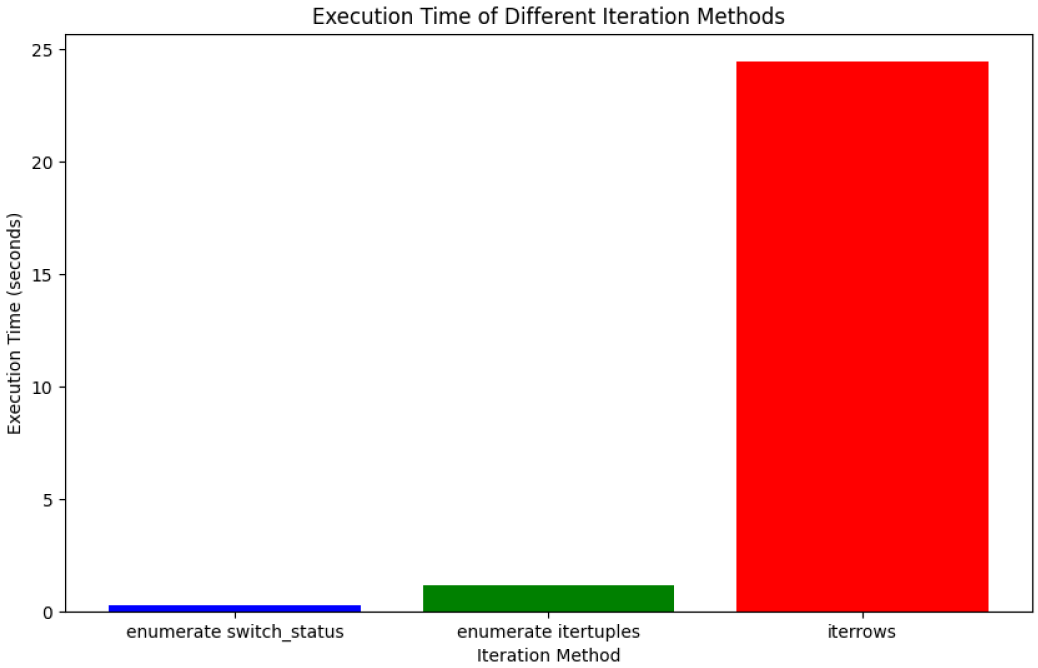

Figure 6: Benchmark comparisons for `df[column]`, df.itertuples(), and df.iterrows().

5 Results

SCS-KNIT yielded successful results after weeks of development. The KNIT toolkit can effectively extract both full and half motions from rows of data entries, graphically represent them, and provide various statistics, both derived and non-derived. SCS-KNIT provides comprehensive statistical plotting for any desired value (column), provided that the value is present. In the context of full motions alone, SCS-KNIT successfully extracted 627 full motions. With these extractions, KNIT can provide operations members data both in JSON and CSV format. While the extraction of motions is the main highlight, the process also included the processing and export of Thermal Ground Support Equipment data, spacecraft thermal data, housekeeping telemetry, and motor current telemetry—all critical to mission success (Figure 7 & 8).

Beyond motion extraction, all modular logic can serve as building blocks for advancing the Sentinel-6 mission or future and ongoing missions, such as the SWOT mission with the Ka-band Radar Interferometer (KaRIn) altimeter, which utilizes the AMR-C instrument. The SCS-KNIT toolkit serves as a foundation for modular instrument operational frameworks architecture and those to come.

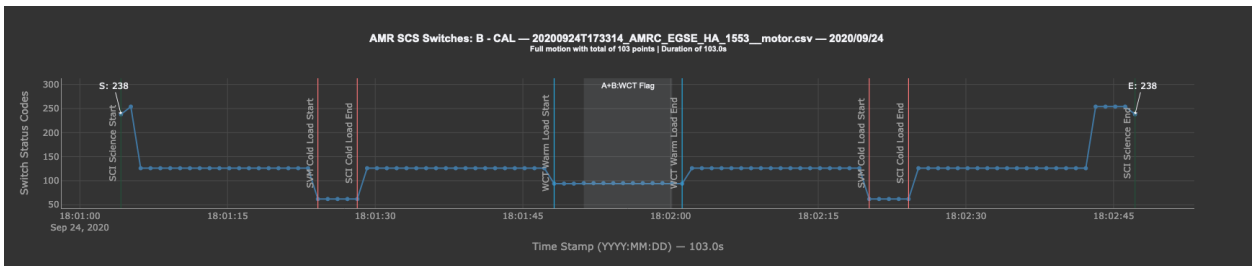

Figure 7: Full motion extraction graphical plots illustrating switch status code state transitions with important state transition markings.

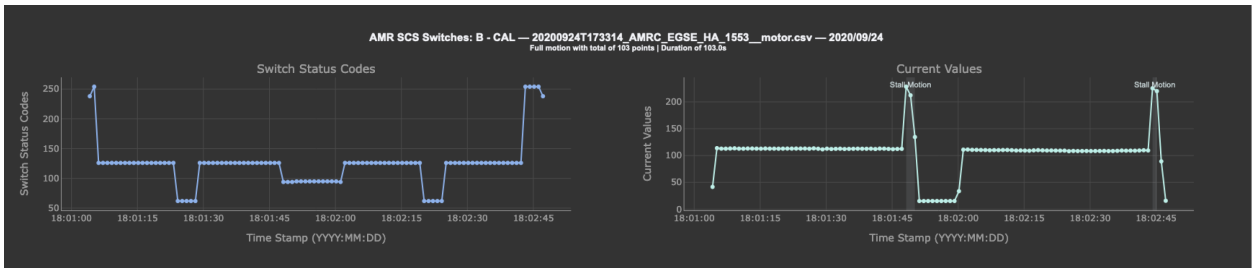

Figure 8: Motion extraction graphical plots illustrating switch status code state transitions and current values.

6 Conclusion

As of now, the implementation phase of SCS-KNIT's framework advancement offers sufficient functionality to fully derive a corpus of SCS motion data. It can extract both full and half motions, generate graphical plots, and produce a master spreadsheet with derived statistics, though it does not include temperature or spacecraft statistics in the sheet. The current progress of the framework meets operational requirements for comparing derived quantities with in-flight calibration telemetry. A Minimum Viable Product (MVP) is available for instrument operations members. However, a fully developed framework should offer complete autonomy and enhanced functionality, including command-line tools, web-based interactive plotting, database connectivity, and automated ingestion into existing databases. Regardless, with the new and improved SCS motion tracking toolkit, SCS-KNIT, the AMR-C instrument holds a software framework that can ensure mission continuity can maintain effective and efficient long-term health monitoring with quick and efficient performance.

7 Acknowledgements

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the JPL Maximizing Student Potential in STEM program and the National Aeronautics and Space Administration (80NM0018D0004). I would like to express my sincere appreciation to my coworkers, mentors, and peers at the laboratory for their continuous support, insightful guidance, and invaluable contributions throughout the entirety of this project.

References

[1] Donlon, C. J., Cullen, R., Giulicchi, L., Vuilleumier, P., Francis, C. R., Kuschnerus, M., Simpson, W., Bouridah, A., Caleno, M., Bertoni, R., Rancaño, J., Pourier, E., Hyslop, A., Mulcahy, J., Knockaert, R., Hunter, C., Webb, A., Fornari, M., Vaze, P., & Tavernier, G. (2021). *The Copernicus Sentinel-6 Mission: Enhanced Continuity of Satellite Sea Level Measurements from Space*. Remote Sensing of Environment, 258, 112395. https://doi.org/10.1016/j.rse.2021.112395

[2] NASA Jet Propulsion Laboratory (JPL). (n.d.). *Sentinel-6 Michael Freilich Satellite*. NASA Jet Propulsion Laboratory (JPL). https://www.jpl.nasa.gov/missions/sentinel-6

[3] *Sentinel-6 Launch Press Kit — Spacecraft and Instruments*. (n.d.). https://www.jpl.nasa.gov/news/press_kits/sentinel-6/mission/spacecraft/

[4] Maiwald, F., Cofield, R., Skalare, A., Statham, S., Ramos, I., Kangaslahti, P., Tripp, K. S., Milligan, L., Bloom, M., Schlecht, E., Nicaise, F., Vo, N., Koch, T., Brown, S., & Kloosterman, J. L. (2017, March 3). *The Advanced Microwave Radiometer – Climate Quality (AMR-C) Instrument for Sentinel-6*. NASA Technical Reports Server (NTRS). https://ntrs.nasa.gov/citations/20210007663

[5] Paulsen, G., Van Dyne, D., Rehnmark, F., Chu, P., & Iskenderian, T. (2020, May 13). *Bearing Anomaly for the Sentinel 6 Supplemental Calibration System*. NASA Technical Reports Server (NTRS). https://ntrs.nasa.gov/citations/20220000871

[6] *WMO OSCAR — Satellite: Sentinel-6B*. (n.d.). https://space.oscar.wmo.int/satellites/view/sentinel_6b

[7] Sharkdp. (n.d.). *GitHub - sharkdp/fd: A Simple, Fast and User-Friendly Alternative to "find."*. GitHub. https://github.com/sharkdp/fd