EF2260 Lab D Hands-on Project Report

Group Number: 4
Group Members: Kirtan Patel, Matteo Ruvolo, Sara Sanchis Climent, Mattia Tadiotto

February 5, 2025

Contents

1 Introduction

Spacecrafts in Low Earth Orbit (LEO) are exposed to various sources of ionizing radiation, including cosmic rays and trapped particles, which can lead to Single Event Upsets (SEUs). SEUs occur when high-energy particles, such as protons or heavier ions, interact with the semiconductor materials in the spacecraft's detectors, causing disruptions or bit flips in the onboard data. These radiation events pose significant risks to the functionality of space systems and require monitoring and analysis.

The MATS (Mesospheric Airglow/Aerosol Tomography and Spectroscopy) satellite is the satellite for a Swedish mission designed to investigate gravity waves in the mesosphere and lower thermosphere. By observing airglow in the O_2 atmospheric band and sunlight scattered by Noctilucent Clouds, the MATS satellite collects data to understand the dynamics of the Earth's middle atmosphere. The satellite utilizes a limb imaging technique combined with tomographic and spectroscopic analysis to observe wave structures and their interactions with atmospheric conditions. Furthermore, as the satellite operates in LEO, it is exposed to high-energy radiation, which can affect the performance of its onboard electronics, making it essential to monitor and analyze potential SEUs that may occur during the mission. The satellite is equipped with a CCD42-10 detector that is exposed to these radiation events and captures high-resolution images, some of which are binned while other are full-frame images. In this project, there are analyzed both types of images to detect and characterize the SEUs.

In the first part of the project, binned images—where each pixel represents a larger area on the detector—are analyzed. This analysis involves detecting individual particle impacts by comparing consecutive frames. A thresholding technique is used to identify regions with significant particle impacts, and the intensity and position of these impacts are recorded. The second part of the project focuses on full-frame images, where single particles and tracks are detected. By analyzing these tracks, the ionization rate along the path can be estimated, and the LET (Linear Energy Transfer) of the impacting particles can be calculated. LET measures the energy deposited by a particle as it traverses the detector, which is useful for identifying the particle type. Notably, heavy ions, exhibit higher LET values than lighter particles like protons.

By combining the analysis of both binned and full-frame images, this project offers an overview of particle impact events on the MATS satellite. The results are compared with SPENVIS (Space Environment, Effects, and Education System) predictions of trapped particle fluxes, enhancing the team's understanding of the radiation environment in LEO and aiding in the assessment of potential impacts on spacecraft electronics.

2 Experimental Setup

This section provides an overview of the experimental setup used in this project to analyze the SEUs in the MATS satellite's CCD detector. The setup includes the equipment used for image acquisition, the types of images involved, and the simulation tools used to assist in processing and analyzing the data.

Experimental Equipment

The equipment utilized includes the MATS satellite's CCD detector and its associated imaging systems. Specifically, the CCD42-10 detector from Teledyne e2v, which will be used for identifying ionization patterns caused by particle impacts in the mesosphere and lower thermosphere.

Objects Under Test

The subjects of analysis are the images captured by the CCD42-10. These images are available in two formats: binned images and full-frame images. The binned images are 44x187 pixel arrays, where each 'superpixel' aggregates the signal from a 40x2 pixel region, representing a larger area of the atmosphere. These images are collected routinely over extended periods and are useful for detecting high-frequency single events, which often occur during brief periods of intense atmospheric activity.

The full-frame images, comprising 2048x511 pixels, offer higher spatial resolution and enable more detailed analysis of individual particle impacts. These images are especially useful for identifying particle tracks within the detector, as they the ionization patterns produced by particle impacts on the CCD can be observed.

2.1 Simulation Tools and Setup

To achieve the results presented in this report, a combination of image processing, simulations, and theoretical modeling was employed. Python was used to develop a custom image processing pipeline, using libraries such as NumPy, Matplotlib, and SciPy for tasks like filtering, thresholding, and extracting relevant features from the image datasets. Additionally, routines were created to calculate ionization rates and LET, apart from enabling the detection and analysis of SEUs within the images. Python-based simulations also modeled ionization patterns in the CCD detector and calculated the angle of particle incidence.

To complement the raw image data and enhance the analysis, simulation tools like SPENVIS were employed to estimate the expected flux of trapped particles, compare observed particle impacts, and validate the results. SPENVIS provided radiation models to simulate the flux of high-energy particles. These simulations allowed for a comparison between experimental results and theoretical predictions, ensuring that the analysis accounted for realistic environmental effects, such as radiation-induced noise in the detectors.

By correlating the SPENVIS results with python simulation, the project was able to identify particle impacts more reliably and gain deeper insights into the MATS satellite's exposure to space radiation and its potential effects on the CCD detector.

3 Method and Sequence of Work

The method was divided into two parts: the analysis of binned images and the analysis of full-frame images. Both analyses involved a combination of image processing, data filtering, and assumptions about particle behavior in the detector. The subsections below explain the steps followed for each part of the project.

3.1 Raw Data

The raw data used in this project consisted of images taken from the MATS satellite together with its precise orbital data. These images were named following a systematic format that encodes the date and time of capture, IR1_YYYYMMDDhhmmss, where **IR1** indicates the imaging channel used (Infrared channel 1); **YYYYMMDD** represents the date in year, month, and day format; and **hhmmss** is the time in hours, minutes, and seconds (UTC) when the image was taken. Thus, an image labeled `IR1_20230819214402` was captured on August 19, 2023, at 21:44:02 UTC.

To support the analysis and enable accurate simulations in SPENVIS, precise orbital information for the MATS satellite was provided. The orbital data followed the standard Two-Line Element (TLE) format. An example of the TLE data is shown below:

```
1 54227U 22147A   22308.79105818 -.00000055  00000+0  00000+0 0  9999
2 54227  97.6565 311.5159 0011681 302.3480 246.6083 14.92786674    05
```

In the TLE format:

- **Line 1:** Contains metadata such as the satellite identifier (`54227U`), launch year and piece (`22147A`), epoch time, and orbital decay terms.

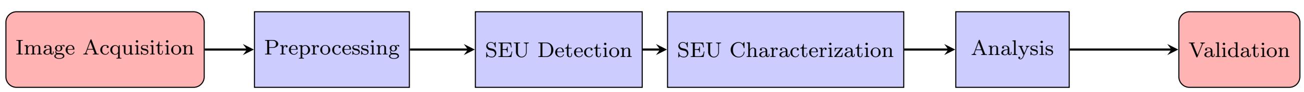
- **Line 2:** Contains the satellite's inclination (`97.6565`), right ascension of the ascending node, eccentricity, argument of perigee, mean anomaly, and mean motion (`14.92786674` revolutions per day).

The TLE data were used as input parameters in the SPENVIS simulation to replicate the MATS satellite's orbital environment accurately. By integrating these orbital elements, the radiation and other conditions surrounding the satellite at the time of image capture were modeled.

3.2 Overview of Methodology

The project workflow, presented in the following flowchart, can be summarized as follows:

- Preprocessing and loading image data (binned and full-frame).
- Identifying SEUs using pixel intensity thresholds.
- Classifying SEUs based on spatial patterns (single-pixel and track-like).
- Estimating parameters, such as energy deposition and ionization rates.
- Visualizing and validating results.



To simplify the workflow, a modular approach was adopted, where each step in the process was broken down into manageable tasks. The assumptions and techniques applied to the binned and full-frame data are described in the following subsections.

3.3 Assumptions in the Code

For both the binned and full-frame images, the following assumptions were made:

1. The CCD detector response was linear (pixel values were proportional to electron count and thus energy deposition).
2. Mean background noise was constant and could be removed through baseline correction.
3. Particle tracks appeared as contiguous bright pixels, and their orientation depended on the particle's incident angle.
4. The threshold used to detect SEUs was fixed across all images and determined using the Dark signal non-uniformity.
5. Events smaller than the threshold noise level were ignored, as they were unlikely to correspond to real SEUs.
6. All events leading a pixel value larger than the threshold were considered SEUs.

3.4 Binned Images Analysis

The binned image analysis involved image processing to identify spatio-temporal peaks in the binned image data to identify SEUs. The process included the following steps:

1. **Image Data Handling**
   Images were loaded into Python using the `numpy` library and the Date and Time was captured using the `datetime` Library. These images were then stored as numpy arrays. It must be noted that the raw pixel values are proportional to the electron counts in the CCD.

2. **Differencing for Temporal Peaks**
   Temporal peaks were calculated using the method described by Chapman [1]. This was implemented by taking a mean of 3 consecutive images and subtracting the mean from the middle image as illustrated in Figures 1 and 2. This way, slowly varying components were removed from the image.

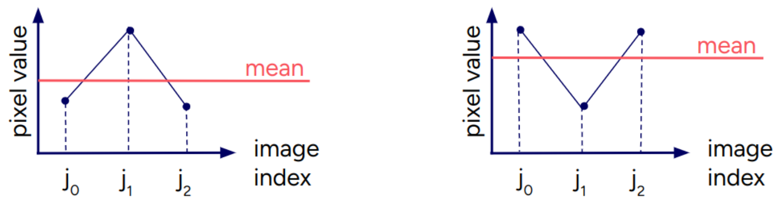


Figure 1: Comparison of the mean with the pixel value in case of a peak [1]

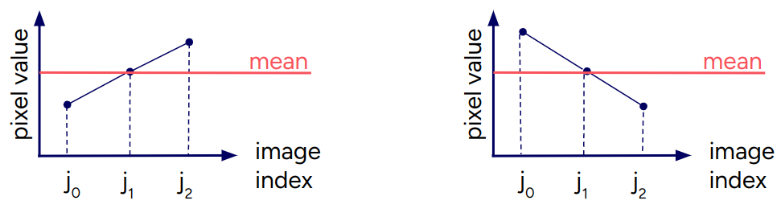


Figure 2: Comparison of the mean with the pixel value in case of no peak [1]

3. **Differencing for Spatial Peaks**
   Spatial peaks were isolated using a combination of windowed median filtering and subtraction. First, a median-filtered version of the image was created, where each pixel was replaced by the median value of its local neighborhood. This process is particularly effective for noise reduction because the median filter is robust to outliers, selectively removing extreme pixel values without distorting the overall structure of the image. Next, the median-filtered image was subtracted from the original unfiltered image, allowing the spatial peaks to be clearly separated from the background. This approach preserves edge details while suppressing noise, ensuring that critical image features remain intact.

4. **Setting the Threshold for SEU Identification**
   The threshold for identifying SEUs was determined by considering the dark-signal non-uniformity (DSNU) of the images. While the dark signal is assumed to be removed through differencing between frames, its inherent variation still persists, and this variation must be accounted for when setting the threshold. To accommodate for this, a 3σ range was applied, establishing a lower limit for the image pixel value threshold, T, of 180 electrons/pixel/second. Given that the exposure time for each image is $t = 5$ s and each binned pixel contains $N = 80$ individual pixels, the

Figure 3: Original image, mean filtered image and median filtered image [2]

threshold for each binned pixel, T_{binned}, is calculated by multiplying the threshold per pixel by the number of pixels in the bin and the exposure time:

$$T_{\text{binned}} = T \times N \times t = 72000 \,(\text{electrons/binned pixel}) \tag{1}$$

Next, we consider the full well capacity of the CCD sensor, which is 100,000 electrons per pixel (or 8,000,000 electrons per binned pixel). Since the pixel values are stored as `uint16` data, with values ranging from 0 to 65535 counts, we can convert the threshold in terms of electrons to a corresponding pixel intensity value. This conversion is done by scaling the binned pixel threshold to the range of the `uint16` format:

$$T_{\text{image}} = T_{\text{binned}} \times \frac{65535}{8000000} \approx 600, (\text{counts/binned pixel}) \tag{2}$$

Finally, to explore how varying the threshold affects SEU detection, the results were plotted for threshold values above this calculated point. This allowed to observe the relationship between an increase in the threshold and a corresponding reduction in the number of SEUs. This trend was further compared to the decrease in the number of energetic particles exceeding a set energy threshold in orbit, providing insight into how changes in threshold impact particle detection in the space environment.

3.5 Full-Frame Images Analysis

The full-frame image analysis involved image processing to identify SEUs in high-resolution CCD images. The process included the following steps:

1. **Image Preprocessing**
   The full-frame images were preprocessed to ensure the data was ready for analysis. First the images were loaded following the same procedure as the binned case and including the `OpenCV` python library. Median filtering was not applied to the full-frame images as it was observed an unintended effect of increasing the number of detected SEUs. It was concluded that while effective for removing noise in binned images, the filter can enhance or distort certain noise elements in the broader, more complex full-frame images, leading to false positives.

2. **Identification of SEUs**
   SEUs were identified based on pixel intensity thresholds within the CCD images. Ionizing particles interacting with the CCD sensor produce energy deposits that manifest as localized increases in pixel intensity. The intensity of a pixel in a CCD image corresponds to the number of electrons collected in that pixel as a result of energy deposition. This relationship arises from the photoelectric conversion properties of the CCD sensor, where energy from ionizing particles liberates

electrons that accumulate in each pixel. The raw pixel values (`uint16` data type) represent this collected charge after analog-to-digital conversion (ADC). The relationship can be expressed as:

$$I_{\text{pixel}} = \frac{Q_{\text{pixel}}}{Q_{\text{full}}} \cdot 65535 \tag{3}$$

where I_{pixel} is the pixel intensity (digital number), Q_{pixel} is the charge collected for the pixel in electrons, and $Q_{\text{full}} = 100,000$ electrons/pixel is the CCD full well capacity. The value 65535 corresponds to the maximum 16-bit output.

Following the same procedure as for the binned images, the pixel intensity threshold for SEU identification was derived from the DSNU. For a full-frame image with an exposure time of $t = 3\,\text{s}$ and a single pixel $N = 1$, the full-frame threshold T_{full} as per Equation 1 results in: $T_{full} = 540$ (electrons/pixel). To convert this threshold to pixel intensity, given Q_{full}electrons/pixel, the threshold value T_{image} is:

$$T_{\text{image}} = T_{\text{full}} \cdot \frac{65535}{100000} \tag{4}$$

resulting in ≈ 350, (counts/pixel). To account for potential uncertainties in DSNU, additional thresholds were evaluated: $\pm 20\%$ variations (280 and 420 (counts/pixel)) and $+50\%$ (525 (counts/pixel)). The aim of this approach is to ensure robust detection across varying noise levels and to allow for SPENVIS simulation comparisons.

Classification of Events: Once detected, SEUs were classified into two types based on their spatial pixel distribution, using connected component analysis from the SciPy library:

(a) **Single-Pixel Events:** Isolated pixels with intensities above the threshold.

(b) **Track-Like Events:** Linear patterns of consecutive bright pixels, with a length > 1 pixel.
Track Angle Calculation: The angle θ of a track was determined using the bounding box of the connected component:

$$\theta = \arctan\left(\frac{\Delta y}{\Delta x}\right)(^\circ), \tag{5}$$

where Δy and Δx are the vertical and horizontal extents of the connected pixels. A threshold of $|\theta| < 20$ $(^\circ)$ was used to identify shallow tracks.

3. **Estimation of Energy Deposition**

The energy deposited by a particle was estimated based on the observed pixel intensities within the SEU region. Assuming a linear relationship between energy deposition and pixel intensity, the energy per pixel is:

$$E_{\text{pixel}} = I_{\text{pixel}} \cdot k, (\text{MeV}), \tag{6}$$

where k is the calibration factor relating pixel intensity to energy in MeV/electron. For the CCD under study, k is approximately 1.35×10^{-3} MeV/electron. [3]

The LET of the particle was approximated as the energy deposited per unit track length. For track-like events, the ionization rate was calculated as:

$$\text{LET} = \frac{E_{\text{total}}}{L}, (\text{MeV/cm}), \tag{7}$$

where L is the track length in centimeters. The track length was determined by multiplying the number of connected pixels by the CCD pixel size ($1.35\,\mu\text{m} = 1.35 \times 10^{-4}\,\text{cm}$). To distinguish between light and heavy ions, an LET threshold of 10 MeV/cm was was chosen to filter out lighter particles, such as protons, which typically have LET values around 1-5 MeV/cm, and to identify heavier ions like alpha particles, carbon and iron (LET >10 MeV/cm).

3.6 Visualization and Validation

The results were visualized to confirm the accuracy of the SEU identification and classification for. Bright pixels and event regions were overlaid on the original images to highlight detected SEUs. Additional plots, such as histograms of SEUs over time and images with heavy particles highlighted, were generated to analyze the distribution of particle impacts. To verify the accuracy and reliability of the experimental results, simulations were performed using SPENVIS, for which the Right Ascension of the Ascending Node was calculated as illustrated in Figure 4 and introduced as input parameter as Local time as seen in Figure 5.

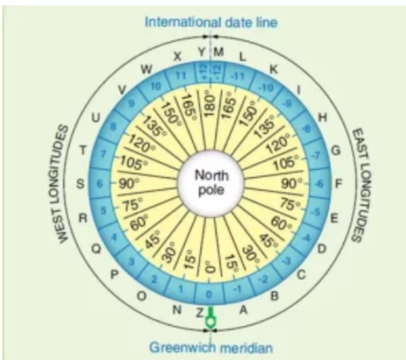


Figure 4: Calculation of time at the Right Ascension of the Ascending Node [4]

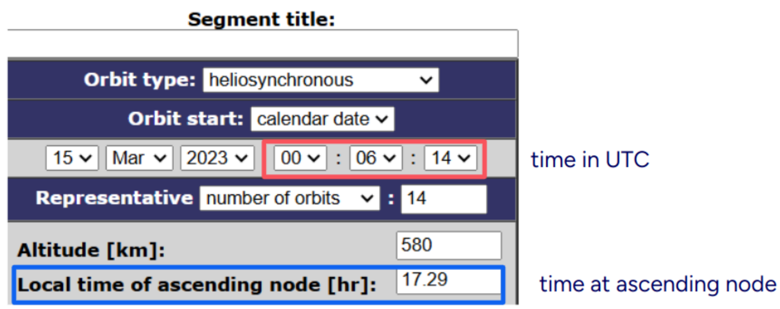


Figure 5: SPENVIS Input Parameters

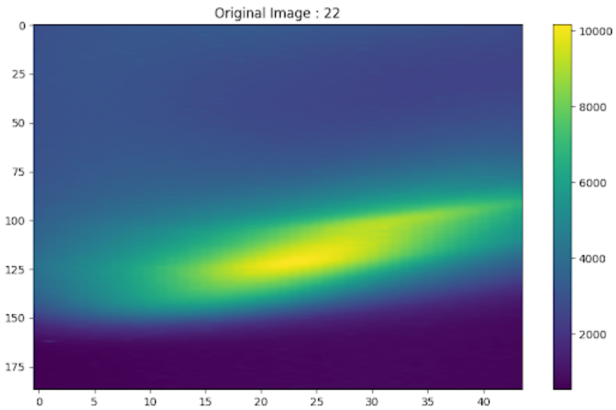
4 Results

The results of the image preprocessing and SEU identification are presented, demonstrating the effectiveness of the methods used. The application of thresholds for SEU detection provided a systematic approach for identifying both single-pixel and track-like events.

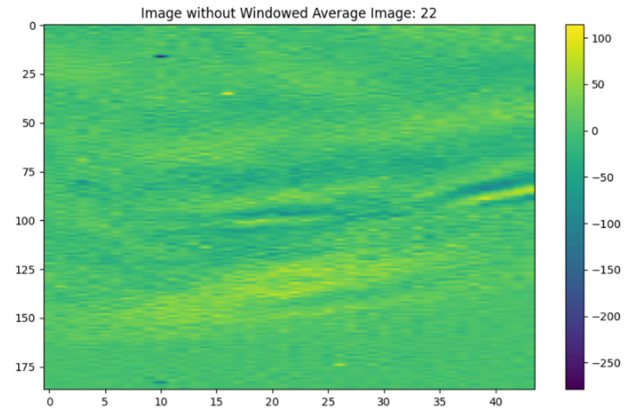
4.1 Binned Images

In the binned image analysis, SEUs counts varied periodically, with a period corresponding to twice the orbit period, suggesting a periodic variation in particle impacts. Figure 6 presents the processing steps of the binned images to illustrate the process of detecting spatio-temporal peaks for SEU identification. Figure 6a displays the original frame. Figure 6b shows the result of subtracting the temporal average (computed as average of the current frame and its neighboring frames) from the original image, isolating temporal peaks by removing non-peak components. Figure 6c applies windowed median filtering to the second image to suppress noise and non-outlier regions. Finally, Figure 6d depicts the difference between Figure 6b and its median-filtered counterpart, Figure 6c, highlighting spatio-temporal peaks. These peaks serve as the basis for SEUs detection leading to the count temporal distribution presented in 7a, and validated with SPENVIS to obtained the Spatial Distribution that can be seen in Figure 7b.

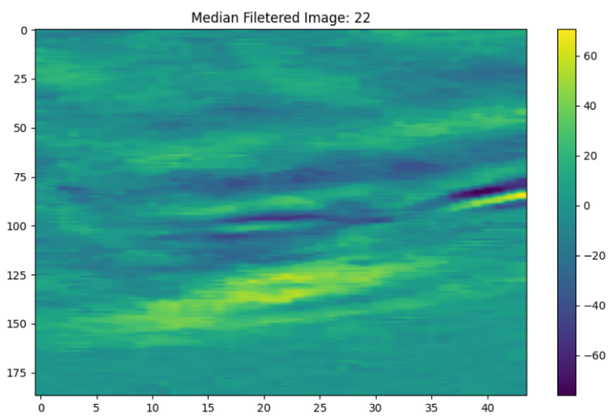
Image Processing



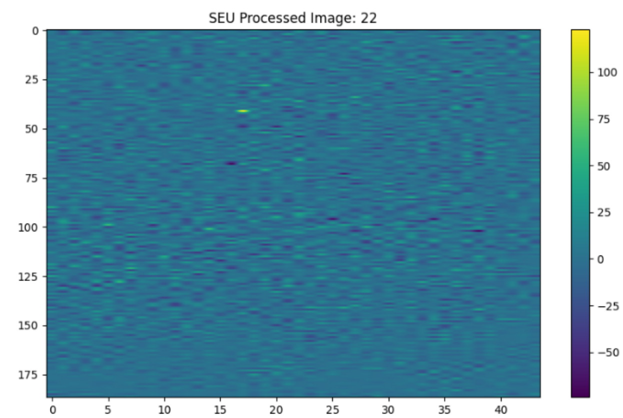
(a) Original image



(b) Image after removal of mean



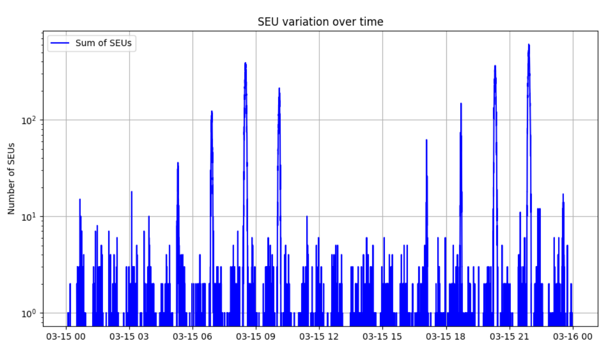
(c) Median Filtered Image



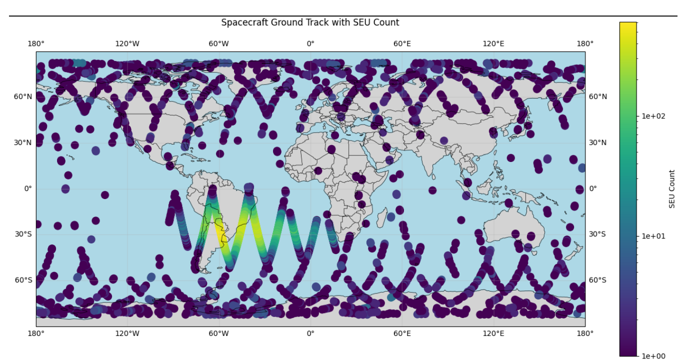
(d) Image without mean and median filtered image

Figure 6: Processing of Image for SEU Counting

SEU Count



(a) Temporal Distribution



(b) Spatial Distribution

Figure 7: Results of Single Event Upsets with Threshold = 600

4.2 Full-frame Images

In this subsection, the full-frame images are presented in their raw (unprocessed) form in Figure 8, with SEUs highlighted in Figure 9, heavy particles highlighted in Figure 10, and the SEU count histograms in Figure 11 for the specified thresholds: 280, 350, 420, and 525. The image *IR1_20230820215019* was selected for this analysis because it demonstrates clear features that allow for comparisons across different thresholds. The variations in SEU detection and heavy particle identification observed in this image are representative of the trends seen in the full dataset. The total number of detected heavy particles and Ionization Rate Range in all images and through the thresholds is presented in Table 1. The SPENVIS simulation results are presented in Figure 12.

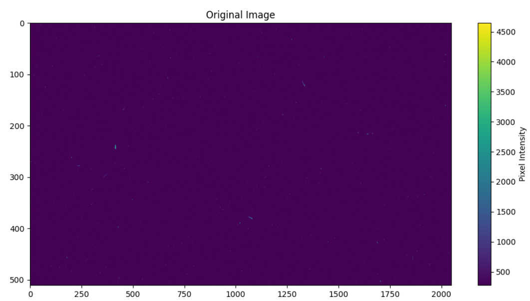
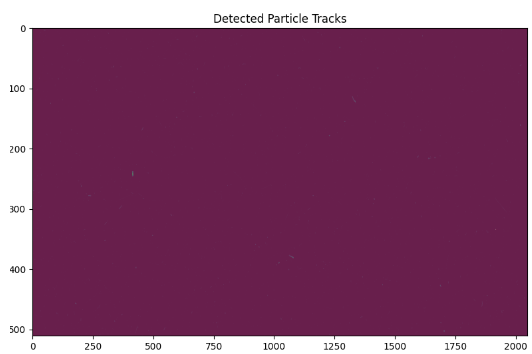
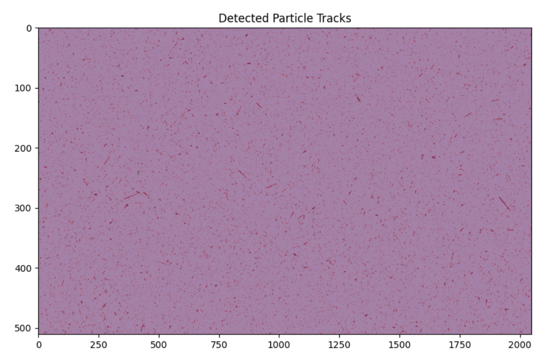


Figure 8: Original full frame image

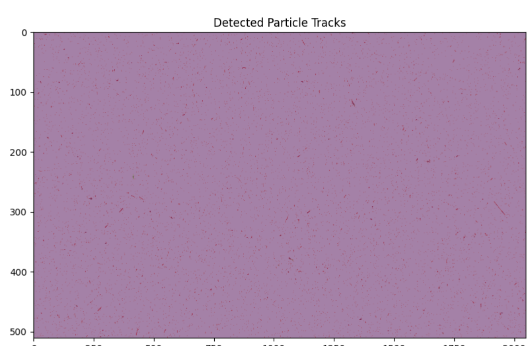
Images SEUs highlighted



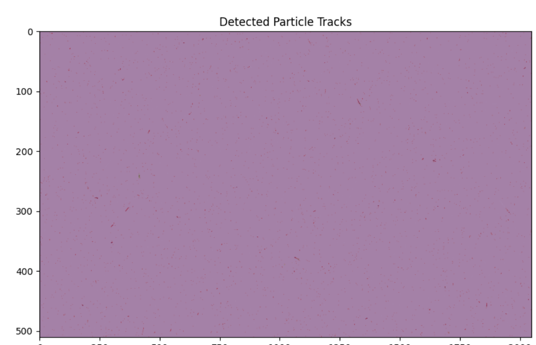
(a) 280 threshold SEUs highlighted image



(b) 350 threshold SEUs highlighted image



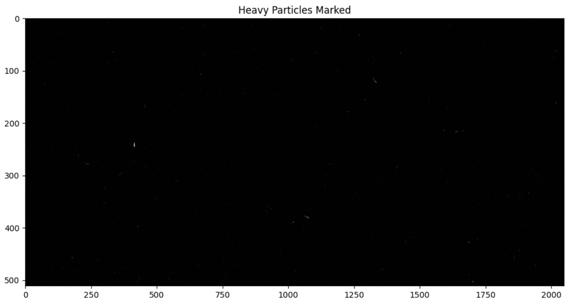
(c) 420 threshold SEUs highlighted image



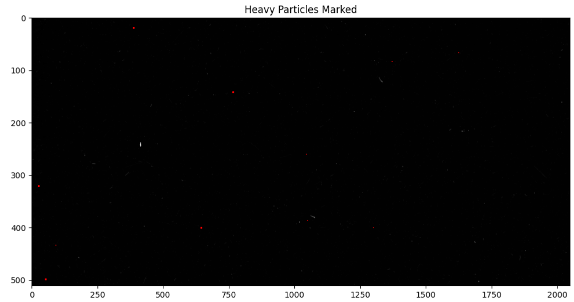
(d) 525 threshold SEUs highlighted image

Figure 9: Processing of Image for SEU highlighting

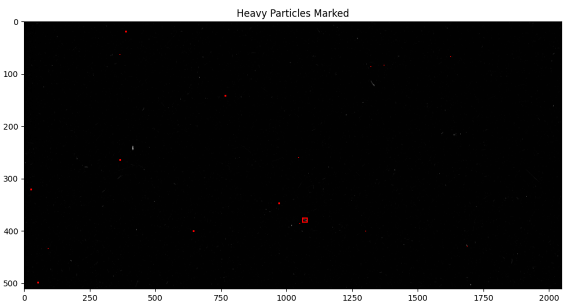
Full frame Images heavy particles highlighted



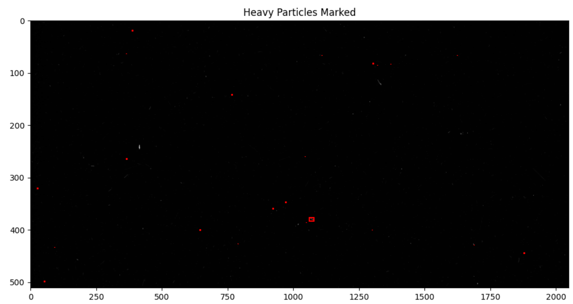
(a) 280 threshold heavy particles highlighted



(b) 350 threshold heavy particles highlighted



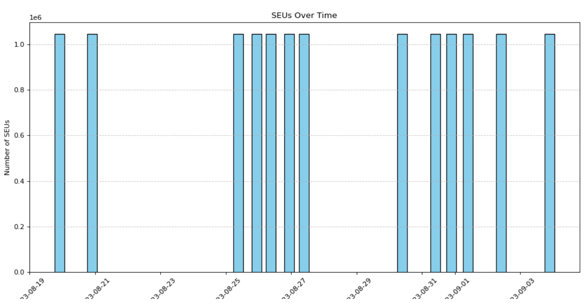
(c) 420 threshold heavy particles highlighted



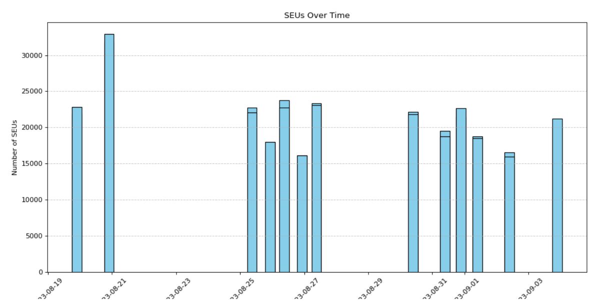
(d) 525 threshold heavy particles highlighted

Figure 10: Processing of Image for heavy particles highlighting

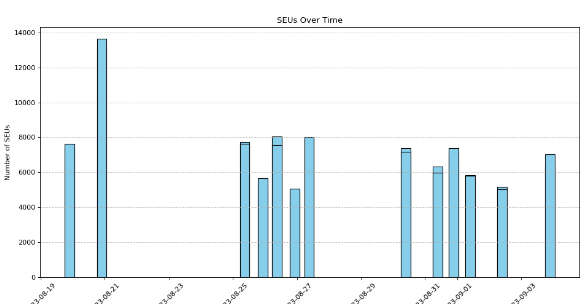
SEUs histogram count



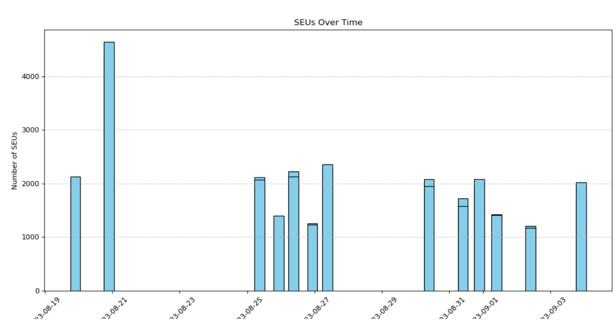
(a) 280 threshold SEUs counting histogram



(b) 350 threshold SEUs counting histogram



(c) 420 threshold SEUs counting histogram



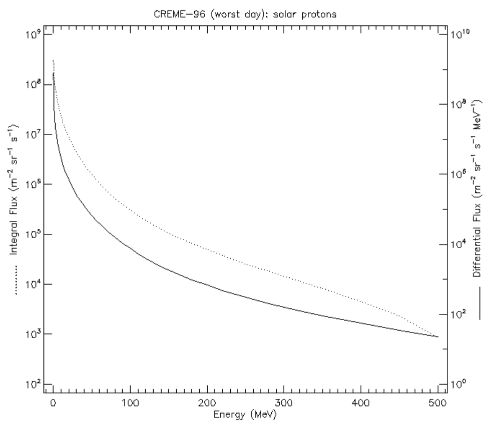
(d) 525 threshold SEUs counting histogram

Figure 11: SEUs Counting Histogram

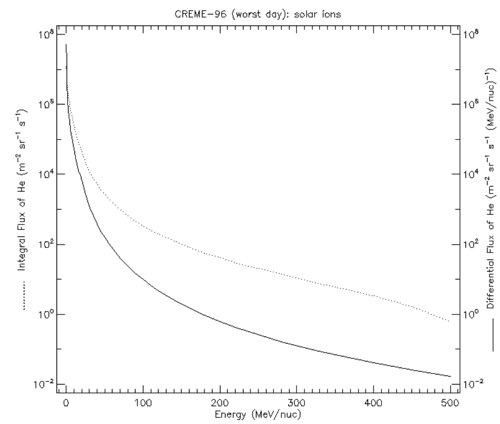
Table 1: Comparison of Heavy Particles Detected and Ionization Rate Range

Threshold	Number of Heavy Particles	Ionization Rate Range (MeV/(cm$^2 \cdot$ s))
280	0	222,797 - 225,595
350	86	260,000 - 2,898,519
420	126	311,852 - 2,898,519
525	278	389,629 - 3,038,519

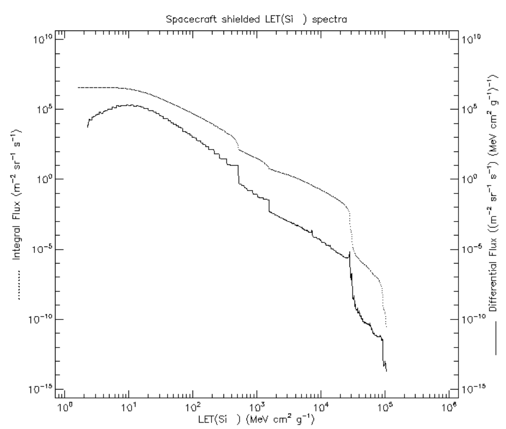
SPENVIS simulation results



(a) Worst Day Solar Protons



(b) Worst Day Solar Ions



(c) Spacecraft Shielded LET SPECTRA

Figure 12: SPENVIS simulation results

5 Discussion

For the binned images analysis, Figure 7 illustrates the results of SEUs with a threshold of 600. The temporal distribution, presented in Figure 7a, shows two frequency components: a higher frequency, approximately twice the orbital frequency, indicating increased SEUs near the poles due to elevated fluxes of high-energy particles in these regions; and a lower frequency, corresponding to SEU spikes as the satellite traverses the South Atlantic Anomaly (SAA), where the Earth's radiation belts result in higher particle flux. The spatial distribution shown in Figure 7b confirms these trends, with SEU activity concentrated near the poles and the SAA, highlighting the link between particle flux and SEU occurrence.

The following subsections present an analysis of the results obtained for different thresholds applied to the full-frame images. The analysis includes comparisons between the thresholds and their corresponding effects on the SEU count, track classifications, and heavy-particle detections.

5.1 General Observations Across Thresholds

Each threshold impacts the number and type of SEUs detected. Considering the results presented in Figures 9 to 11 and Table 1, it can be seen a clear relationship between the threshold and number of SEUs and heavy particles:

- **Threshold 280:** Detects the highest number of SEUs. Using a low threshold leads to including significant noise, which is misidentified as SEUs. No heavy particles were detected. The SEU count is likely overestimated, with faint or low-energy pixel variations detected.
- **Threshold 350:** Provides a good balance between sensitivity and noise suppression. Compared to 280, noise artifacts are reduced, and detected SEUs are more reliable. A significant increase in heavy-particle detection (86) is observed. This threshold retains both single-pixel and track-like events while minimizing false positives.
- **Threshold 420:** Less SEUs are detected, as low-intensity signals are filtered out. The SEUs predominantly correspond to higher-energy events, and noise is significantly reduced. Track-like features become sparse but remain well-defined. The number of heavy particles increases to 126 and ionization rates for detected particles remain within a consistent range.
- **Threshold 525:** This threshold is the most stringent and identifies only the brightest SEUs. The highest number of heavy particles (278) is detected here, indicating only the brightest and most energetic tracks are retained, which corresponds to higher ionization rates. However, some valid low-energy SEUs might be excluded, having the SEU count drastically reduced.

The results demonstrate that as the threshold increases, the number of detected SEUs decreases, with only high-intensity events remaining. Overall, threshold 350 strikes the best compromise between noise reduction and sensitivity to real heavy-particle events. Higher thresholds (420 and 525) provide more selective detection but might exclude lower-intensity events.

5.2 Comparison of SPENVIS Results

The results from the SPENVIS simulation indicate a strong dependence of flux on particle energy: In Figure 12a, the flux decreases exponentially as energy increases. High-energy protons are relatively rare, which explains the decrease in detected heavy particles for lower thresholds. Analysing Figure 12b, helium ions dominate at lower energies, and their flux decreases for energies beyond $\sim$100 MeV/nuc. Finally, from Figure 12c the Shielded LET spectra can be studied. LET flux follows a steep decline at higher LET values (100 MeV$\cdot$cm^2/g) confirming the stricter thresholds (e.g. 525) are detecting only the rare, high-LET events while excluding the lower-energy contributions.

Therefore, SPENVIS data validates the behavior observed for the defined thresholds:

- **Threshold 350**: Detecting 86 heavy particles is consistent with SPENVIS flux distributions for moderate-energy protons and helium ions. The ionization energy range aligns with regions where LET flux is significant.
- **Threshold 525**: While detecting 278 heavy particles, this threshold aligns with the high-LET region observed in the SPENVIS spectra. However, the flux in this region is orders of magnitude lower, corresponding to rarer high-energy events.

5.3 Uncertainties and Validity of Assumptions

The results obtained are subject to several uncertainties and rely on key assumptions, which influence the overall credibility of the findings:

- **Intensity and Threshold Selection:** The primary threshold for SEU detection (350) was calculated based on DSNU assumptions. However, variations in background noise or detector calibration may introduce errors. The alternative thresholds (280, 420, and 525) were included to explore the sensitivity of results to threshold selection.

- **Pixel Intensity Interpretation:** Assuming pixel intensity is proportional to energy deposition is due to the linear response of the CCD detector. However, saturation effects or variations in CCD sensitivity might cause deviations, particularly for high-intensity events.

- **Angle Calculation for Track-Like Events:** The shallow angle threshold (20°) was chosen to exclude steep tracks unlikely to correspond to near-perpendicular particle trajectories. However, this fixed threshold introduces an uncertainty, as some valid tracks at slightly larger angles may be excluded.

- **LET Threshold for Heavy Particles:** The LET threshold of 10 MeV/cm was selected to identify particles heavier than protons. This threshold is based on typical LET values for protons ($1 - 10$ MeV/cm) and heavier ions like alpha particles and carbon nuclei (> 10 MeV/cm). While reasonable, this assumption could be refined with more precise calibration.

- **Background Noise:** Background noise from dark currents and cosmic background radiation can obscure faint SEUs.

- **Detector Artifacts:** Damaged pixels or permanent defects in the CCD detector could result in false positives. For instance, pixels previously damaged by highly energetic particles might produce consistent high-intensity values in subsequent images, which was observed comparing the heavy particles highlighted images taken for the same threshold at different times.

5.4 Suggestions for Future Improvements

The following improvements are suggested to enhance the accuracy and robustness of the analysis:

- **Threshold Optimization:** Apply a Gaussian fit to the SEU-processed image, exploiting the fact that instrument noise typically follows a Gaussian distribution, to identify outliers that correspond to SEUs for further analysis and mitigation.

- **Adaptive Threshold Calibration:** Implement dynamic thresholding techniques that adapt to variations in background noise across images. This would improve SEU detection consistency without over-relying on static thresholds.

- **Damaged Pixel Identification:** Develop algorithms to identify and exclude permanently damaged pixels that consistently show high intensity values, thus reducing false positives.

6 Conclusion

This project analyzed CCD42-10 detector data from the MATS satellite to detect and characterize SEUs caused by particle impacts in the mesosphere and lower thermosphere. Using image processing, theoretical modeling, and space environment simulations, the project highlighted the dynamic nature of radiation-induced SEUs and demonstrated the ability to identify heavy-ion tracks in full-frame images.

The main conclusions include the following:

- **SEU Detection in Binned Images:** An image processing pipeline isolated spatio-temporal peaks to identify SEUs, revealing periodic variations linked to the satellite's orbital environment. The SEU frequency correlated with expected particle energy distributions, with the SAA as the primary source.
- **SEU Detection in Full-Frame Images:** Full-frame images enabled classification of SEUs into single-pixel and track-like events. Ionization rates and LET distinguished heavy ions from protons. Threshold selection was critical: lower thresholds (e.g., 280) overestimated SEUs, while higher thresholds (e.g., 525) excluded lower-intensity tracks, with 350 counts per pixel providing an optimal balance.
- **Simulation and Validation:** SPENVIS simulations accurately modeled the radiation environment, validating experimental results and flux estimations. The SPENVIS simulation confirmed the flux of particles contributing to SEUs decreases with increasing energy and LET values. The comparison showed that Threshold 350 aligned well with SPENVIS's moderate-energy flux profiles, offering a balanced detection approach.

In conclusion, the project integrated image processing, simulations, and theoretical analysis to detect and characterize SEUs, providing a framework for understanding space radiation effects on CCD detectors.

7 Group Member Contributions

Member	**Lab**	**Presentation**	**Review**	**Report**
Kirtan Patel	Binned images simulations and coding	Binned Images and Conclusions	Individual	Binned Images method and results, Discussion and conclusion
Matteo Ruvolo	SPENVIS Full Frame	-	Individual	Discussion and Conclusion
Sara Sanchis Climent	Full frame images simulations and coding	Introduction, Full Frame Images and Conclusion	Individual	Introduction, Setup, Method General, Full Frame Images method and results, Discussion and Conclusion
Mattia Tadiotto	SPENVIS Binned	-	Individual	Discussion and Conclusion

Table 2: Group members' contributions

References

[1] G. H. Chapman *et al.*, "Single event upsets and hot pixels in digital imagers," in *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*. Amherst, MA, USA: IEEE, Oct. 2015, pp. 41–46.

[2] P. Murugan, "Decision based adaptive gradient mean filter(dbagm)," 2018.

[3] e2v technologies, "Ccd42-10 datasheet," 2016. [Online]. Available: http://www.e2v.com/resources/account/download-datasheet/3747

[4] G. Cameron, "Calculating time and date," https://cameroongcerevision.com/calculating-time-and-date/, 2024, accessed: Dec. 13, 2024.

A Appendices

Binned Images Code

```
import numpy as np
from PIL import Image
from datetime import datetime, timedelta
import os
import matplotlib.pyplot as plt
from scipy.ndimage import median_filter, uniform_filter
import matplotlib.colors as colors
import gc

def seu_image_processing(images, image_index, window_size=3, filter_size=3,
    new_min=-1, new_max=1):
    """
    Comprehensive image processing function with multiple operations.
    Args:
        images (numpy.ndarray): Input 3D array of images
        window_size (int): Size of sliding window for average calculation
        filter_size (int): Size of median filter
        new_min (float): Minimum value for scaling
        new_max (float): Maximum value for scaling
    Returns:
        numpy.ndarray: Processed and scaled image array
    """
    # Validate input
    if images.ndim != 3:
        raise ValueError("Input must be a 3D numpy array with shape (n, a,
            b).")
    if images.shape[0] < window_size:
        raise ValueError(f"Input must have at least {window_size} images.")

    # Compute sliding window average
    windowed_avg = np.array([
        np.mean(images[i:i+window_size], axis=0)
        for i in range(images.shape[0] - window_size + 1)
    ])

    print("window average computed")

    '''
    plt.figure(figsize=(10, 6))
    plt.imshow(windowed_avg[image_index], cmap='viridis', aspect='auto')
    plt.colorbar()
    plt.title(f"Windowed Average Image: {image_index}")
    plt.show()
    '''

    # Slice original images
    images = images[1:-1]

    # Compute element-wise difference
    images = images - windowed_avg

    '''
```

```
    plt.figure(figsize=(10, 6))
    plt.imshow(images[image_index], cmap='viridis', aspect='auto')
    plt.colorbar()
    plt.title(f"Image without Windowed Average Image: {image_index}")
    plt.show()
    '''

    print("image difference calculated")

    # freeing space to avoid the process from getting killed
    del windowed_avg
    # Run garbage collection manually
    gc.collect()

    # Apply median filtering
    median_filtered = np.array([
        median_filter(img, size=filter_size)
        for img in images
    ])

    '''
    # Apply mean filtering
    median_filtered = np.array([
        uniform_filter(img, size=filter_size)
        for img in images
    ])
    '''

    print("difference median filtered")

    '''
    plt.figure(figsize=(10, 6))
    plt.imshow(median_filtered[image_index], cmap='viridis', aspect='auto')
    plt.colorbar()
    plt.title(f"Mean Filetered Image: {image_index}")
    plt.show()
    '''

    # Extract noise by taking difference between original and filtered
    images = images - median_filtered

    # freeing space to avoid the process from getting killed
    del median_filtered
    # Run garbage collection manually
    gc.collect()

    print("noise extracted")

    '''

    # ONLY SCALING
    # Scale each image independently
    scaled_noise = np.zeros_like(images, dtype=float)
    for i in range(images.shape[0]):
        img = images[i]

```

```
        # Skip scaling if image is constant
        if np.min(img) == np.max(img):
            scaled_noise[i] = np.zeros_like(img)
        else:
            # Scale to specified range for each individual image
            scaled_noise[i] = ((img - np.min(img)) /
                                (np.max(img) - np.min(img))) * (new_max - new_min) + new_min

    # freeing space to avoid the process from getting killed
    del images
    # Run garbage collection manually
    gc.collect()

    print("scaled noise calculated")
    '''

    # THRESHOLDING
    # Set the threshold for SEU detection
    threshold = 250

    # Apply the threshold to create a binary matrix
    binary_images = (abs(images) >= threshold).astype(np.int8)


    # some images have more 1's than zeros.
    # This will be the case when the SEUs cause a dip in intensity, rather than spike
    # thus, we invert images with more 1's than zeros, to easily count the seu_sums
    # the number of SEUs for all images then simply corresponds to the sum of elements of the images

    # Iterate through each image and invert if 1s are the majority
    for i in range(binary_images.shape[0]):
        image = binary_images[i]  # Select the i-th image

        # Count the number of 1s
        num_ones = np.sum(image)

        # If 1s are more than 0s, invert the image
        if num_ones > (image.size / 2):
            binary_images[i] = 1 - image  # Invert the image (1 -> 0, 0 -> 1)

    return binary_images

def save_binary_images_with_names(matrix, datetimes, output_dir, format="png"):
    """
    Saves each (1, a, b) slice of a binary (n, a, b) matrix as a black-and-white image,
    using corresponding names from a string array as filenames.

    Parameters:
        matrix (numpy.ndarray): Input 3D binary matrix of shape (n, a, b), values 0 or 1.
```

```
        filenames (list of str): Array of n filenames (without extensions).
        output_dir (str): Directory to save the images.
        format (str): Image format, e.g., "png".
    """
    # Ensure the matrix is a NumPy array
    matrix = np.asarray(matrix)

    # Check if the input is 3D
    if matrix.ndim != 3:
        raise ValueError("Input matrix must be a 3D array of shape (n, a, b).")

    # Check if the matrix is binary
    if not np.all((matrix == 0) | (matrix == 1)):
        raise ValueError("Input matrix must only contain binary values (0 and 1).")

    # Check if filenames match the number of slices
    if len(datetimes) != matrix.shape[0]:
        raise ValueError("Length of filenames array must match the number of slices in the matrix.")

    # Create output directory if it doesn't exist
    os.makedirs(output_dir, exist_ok=True)

    # Iterate over each (a, b) slice in the matrix and corresponding filename
    for i in range(matrix.shape[0]):
        # Convert the binary slice to a Pillow image in mode '1' (1-bit pixels)
        image = Image.fromarray(matrix[i].astype(np.uint8) * 255)  # Scale 0/1 to 0/255
        image = image.convert('1')  # Convert to 1-bit pixels (black-and-white)

        # Convert datetime to YYYYMMDDHHMMSS format
        filename = datetimes[i].strftime("%Y%m%d%H%M%S") + f".{format}"

        # Save the image with the corresponding filename
        image.save(os.path.join(output_dir, filename))

    print(f"Saved {matrix.shape[0]} binary images to {output_dir}")

def calculate_sums(matrix):
    """
    Calculate the sum of elements for each (1, a, b) slice in an (n, a, b) binary matrix.

    Parameters:
        matrix (numpy.ndarray): Input 3D binary matrix of shape (n, a, b), values 0 or 1.

    Returns:
        numpy.ndarray: A 2D array of shape (n, 1), containing the sum of elements for each slice.
    """
    # Ensure the matrix is a NumPy array
```

```
    matrix = np.asarray(matrix)

    # Check if the input is 3D
    if matrix.ndim != 3:
        raise ValueError("Input matrix must be a 3D array of shape (n, a, b).")

    # Check if the matrix is binary
    if not np.all((matrix == 0) | (matrix == 1)):
        raise ValueError("Input matrix must only contain binary values (0 and 1).")

    # Calculate the sum of each (a, b) slice
    slice_sums = np.sum(matrix, axis=(1, 2), keepdims=True)

    return slice_sums


# index of image whose progression will be plot throughout
image_progess_index = 22


# Define the folder path containing the .bin files
folder_path =
    '/home/kirtan/local-repository/KTH-EF2260-Space-Environment-and-Spacecraft-Engineerin
     # Replace with the correct path
datetime_array_filepath = folder_path+'date-time.npy'
image_data_filepath = folder_path+'image-data.npy'

dates_array = np.load(datetime_array_filepath, allow_pickle=True)
images_array = np.load(image_data_filepath, allow_pickle=True)

# very important to get a sensible value of differences
images_array = images_array.astype(np.int16)

print("image array imported")

'''
plt.figure(figsize=(10, 6))
plt.imshow(images_array[image_progess_index+1], cmap='viridis', aspect='auto')
plt.colorbar()
plt.title(f"Original Image : {image_progess_index}")
plt.show()
'''

seu_identifiable_images = (seu_image_processing(images_array,
    image_progess_index)).astype(np.int8)
seu_identifiable_dates = dates_array[1:-1]

# freeing space to avoid the process from getting killed
del dates_array
del images_array
# Run garbage collection manually
gc.collect()

'''
```

```
# TO SAVE IMAGEs
output_directory = "/home/kirtan/github/KTH-EF2260-Space-Environment-and-Spacecraft-Engineering/image_da

# Save the images
save_binary_images_with_names(seu_identifiable_images, seu_identifiable_dates, output_directory, format="png")
'''

# Calculate the sums for each (1, a, b) slice
seu_sums = calculate_sums(seu_identifiable_images)

# Flatten sums to match x_values
seu_sums = seu_sums.flatten()


# Plot
plt.figure(figsize=(8, 6))
plt.plot(seu_identifiable_dates, seu_sums, linestyle='-', color='b', label="Sum of SEUs")
plt.yscale('log')
plt.xlabel("DateTime")
plt.ylabel("Number of SEUs")
plt.title("SEU variation over time")
plt.grid(True)
plt.legend()
plt.show()


# BINNING THE DATA OVER TIME
bin_size_seconds = 900

# Initialize bins
start_time = seu_identifiable_dates[0]
end_time = seu_identifiable_dates[-1]
bin_size = timedelta(seconds=bin_size_seconds)
current_bin_start = start_time
binned_dates = []
binned_sums = []

while current_bin_start <= end_time:
    # Determine the end of the current bin
    current_bin_end = current_bin_start + bin_size

    # Find indices of sums within the current bin
    in_bin = (seu_identifiable_dates >= current_bin_start) & (seu_identifiable_dates < current_bin_end)

    # Sum SEUs in the current bin
    bin_sum = np.sum(seu_sums[in_bin])
    binned_dates.append(current_bin_start)
    binned_sums.append(bin_sum)

    # Move to the next bin
    current_bin_start = current_bin_end

```

```

# Plot the binned data as a bar chart
plt.figure(figsize=(10, 6))
plt.bar(binned_dates, binned_sums, width=0.01, color='b', label="Binned SEUs")  # Bar chart with width adjusted for readability
plt.yscale('log')
plt.xlabel("DateTime")
plt.ylabel("Number of SEUs")
plt.title("SEU Variation (Binned Every 15 Minutes)")
plt.grid(True)
plt.legend()
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()


'''
using averaging over 3 images, we can not use the first and the second image.
Thus, for initial number of images = N, the SEU identifiable images are N-2
thus, the date-time information can be matched with the SEU identifiable images
by removing the first and last elements
'''

'''

plt.figure(figsize=(10, 6))
plt.imshow(seu_identifiable_images[10], cmap='viridis', aspect='auto')
plt.colorbar()
plt.title(f"SEU Processed Image: {image_progess_index}")
plt.show()

plt.figure(figsize=(10, 6))
plt.imshow(np.log(abs(seu_identifiable_images[100])), cmap='viridis', aspect='auto')
plt.colorbar()
plt.title(f"SEU Processed Image: {100}")
plt.show()

plt.figure(figsize=(10, 6))
plt.imshow(np.log(abs(seu_identifiable_images[1000])), cmap='viridis', aspect='auto')
plt.colorbar()
plt.title(f"SEU Processed Image: {1000}")
plt.show()

plt.figure(figsize=(10, 6))
plt.imshow(np.log(abs(seu_identifiable_images[10000])), cmap='viridis', aspect='auto')
plt.colorbar()
plt.title(f"SEU Processed Image: {10000}")
plt.show()
'''

'''
# FOR ONLY SCALED IMAGES
```

```

plt.figure(figsize=(10, 6))
im = plt.imshow(seu_identifiable_images[10], cmap='viridis', aspect='auto', norm=colors.TwoSlopeNorm(vmin=-1, vcenter=0, vmax=1))
cbar = plt.colorbar(im)
cbar.set_ticks([-1, 0, 1])
cbar.set_ticklabels(['-1', '0', '1'])
plt.title(f"SEU Processed Image: {10}")
plt.show()
'''

import os
import glob
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt

# Function to extract datetime from filename (assuming format: 'IR1_YYYYMMDDHHMMSS')
def extract_datetime_from_filename(filename):
    basename = os.path.basename(filename)  # Extracts just the filename (no path)
    datetime_str = basename[4:].replace('.bin', '')  # e.g., '20230315000614'
    return datetime.strptime(datetime_str, '%Y%m%d%H%M%S')

# Define the folder path containing the .bin files
folder_path = '/home/kirtan/github/KTH-EF2260-Space-Environment-and-Spacecraft-Engineering/image_da  # Replace with the correct path

# List all .bin files in the folder
bin_files = glob.glob(os.path.join(folder_path, '*.bin'))

# Check if there are any .bin files
if not bin_files:
    print("No .bin files found in the folder.")
    exit()

# Sort files based on the datetime extracted from the filename
bin_files_sorted = sorted(bin_files, key=extract_datetime_from_filename)

# Function to read and store images and dates into separate arrays
def load_images_and_dates(sorted_bin_files):
    dates = []  # List to store the dates extracted from filenames
    images = []  # List to store image data (as numpy arrays)

    # Loop through each .bin file in chronological order
    for bin_file in sorted_bin_files:
        # Extract the date from the filename and append it to the dates list
        capture_datetime = extract_datetime_from_filename(bin_file)
        dates.append(capture_datetime)

        # Read the binary data and reshape it into the expected image shape
        with open(bin_file, 'rb') as file:
            image_data = np.fromfile(file, dtype=np.uint16, count=187 * 44).reshape(187, 44)
```

```
            images.append(image_data)

    # Convert lists to numpy arrays
    dates_array = np.array(dates)
    images_array = np.array(images)

    return dates_array, images_array



# Define the folder path containing the .bin files
folder_path =
    '/home/kirtan/local-repository/KTH-EF2260-Space-Environment-and-Spacecraft-Engineerin
     # Replace with the correct path

# List all .bin files in the folder
bin_files = glob.glob(os.path.join(folder_path, '*.bin'))

# Check if there are any .bin files
if not bin_files:
    print("No .bin files found in the folder.")
    exit()

# Sort files based on the datetime extracted from the filename
bin_files_sorted = sorted(bin_files, key=extract_datetime_from_filename)

# Load images and dates into separate arrays (in chronological order)
dates_array, images_array = load_images_and_dates(bin_files_sorted)

(data_length, image_width, image_height) = images_array.shape
# print(data_length)
# print(image_width)
# print(image_height)

'''
# Optionally, display the first image (now in chronological order)
plt.figure(figsize=(10, 6))
plt.imshow(images_array[0], cmap='viridis', aspect='auto')
plt.colorbar()
plt.title(f"First Image (Chronologically):
    {os.path.basename(bin_files_sorted[0])}")
plt.show()
'''

# Save the date_time array in .npy format (in chronological order)
np.save('date_time.npy', dates_array)
# Save the image_data array in .npy format (in chronological order)
np.save('image_data.npy', imag


import numpy as np
import matplotlib.pyplot as plt

# Define the folder path containing the .bin files
folder_path =
    '/home/kirtan/github/KTH-EF2260-Space-Environment-and-Spacecraft-Engineering/image_da
```

```python
image_name = 'IR1_20230315000614.bin'

filepath = folder_path + image_name

# Read binary data from the file
ff = open(filepath, 'rb')
image = np.fromfile(ff, dtype=np.uint16, count=187*44).reshape(187, 44)
ff.close()

plt.figure(figsize=(10, 6))
plt.imshow(image, cmap='viridis', aspect='auto')
plt.colorbar()
plt.show()
```

Full frame Images Code

```python
import csv, datetime, matplotlib, os

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from scipy.ndimage import label, find_objects
from collections import defaultdict
from matplotlib.patches import Rectangle
from datetime import datetime

matplotlib.use('Agg')

# Constants for LET comparison
PIXEL_SIZE_CM = 1.35e-3  # Pixel size in cm
ENERGY_PER_SEU_MEV = 1e-5  # Energy deposited per SEU in MeV

def load_full_frame_image(filepath):
    with open(filepath, 'rb') as file:
        image = np.fromfile(file, dtype=np.uint16, count=2048 *
            511).reshape(511, 2048)
    return image

def detect_particle_impacts(image, threshold=280):
    """
    Detect particle impacts in a full-frame image.
    """
    binary_image = (image > threshold).astype(np.int8)
    return binary_image


def analyze_tracks(binary_image, original_image, angle_threshold=20):
    """
    Analyze particle tracks and compute ionization rate and LET.
    Only include tracks at shallow angles with respect to the CCD chip.
    """
    labeled_image, num_features = label(binary_image)
    track_data = []

    for i in range(1, num_features + 1):
```

```python
        track_pixels = np.argwhere(labeled_image == i)  # List of (y, x) coordinates
        intensity = np.sum(original_image[labeled_image == i])
        length = len(track_pixels)

        # Calculate angle
        y_coords, x_coords = zip(*track_pixels)
        angle = np.rad2deg(np.arctan2(max(y_coords) - min(y_coords), max(x_coords) - min(x_coords)))

        # Skip tracks not meeting shallow angle criteria
        if abs(angle) > angle_threshold:
            continue

        # Calculate ionization rate and LET
        # Track length in pixels
        length_pixels = len(track_pixels)
        # Track length in cm
        length_cm = length_pixels * PIXEL_SIZE_CM
        # Ionization rate
        ionization_rate = intensity / length_cm if length_cm > 0 else 0
        # LET estimation
        let = ionization_rate * ENERGY_PER_SEU_MEV

        track_data.append({
            'intensity': intensity,
            'length': length,
            'angle': angle,
            'ionization_rate': ionization_rate,
            'LET': let,
            'bounding_box': find_objects(labeled_image == i)[0]})

    return track_data

# Convert track data to a DataFrame

def identify_heavy_particles(tracks, let_threshold=10):
    """
    Identify tracks with high LET, indicative of heavy particles.
    """
    heavy_particles = [track for track in tracks if track['LET'] > let_threshold]
    print(f"\nParticles with LET > {let_threshold} MeV/cm (Possible heavy particles):")
    for i, track in enumerate(heavy_particles):
        print(f"Track {i+1}: LET={track['LET']:.3e} MeV/cm, Ionization Rate={track['ionization_rate']:.2f} SEUs/cm")
    return heavy_particles

def extract_timestamp(filename):
    """
    Extract timestamp from filename.
    """
    timestamp = filename.split('_')[1][:14]
    return datetime.strptime(timestamp, "%Y%m%d%H%M%S")

```

```python
def mark_heavy_particles(original_image, heavy_particles, output_path):
    fig, ax = plt.subplots(figsize=(12, 6))
    ax.imshow(original_image, cmap='gray', aspect='auto')
    for track in heavy_particles:
        bbox = track['bounding_box']
        if bbox:
            yslice, xslice = bbox
            rect = Rectangle((xslice.start, yslice.start),
                             xslice.stop - xslice.start,
                             yslice.stop - yslice.start,
                             linewidth=1.5, edgecolor='red', facecolor='none')
            ax.add_patch(rect)
    plt.title("Heavy Particles Marked")
    plt.savefig(output_path)
    plt.close()


def save_original_image(original_image, output_path):
    """
    Save the original image in PNG format for comparison.
    """
    plt.figure(figsize=(12, 6))
    plt.imshow(original_image, aspect='auto')
    plt.colorbar(label="Pixel Intensity")
    plt.title("Original Image")
    plt.savefig(output_path)
    plt.close()


def plot_detected_tracks(original_image, binary_image, tracks, output_path,
    csv_path):
    """
    Plot original image and annotate detected tracks.
    """
    plt.figure(figsize=(12, 6))
    plt.imshow(original_image, cmap='viridis', aspect='auto', alpha=0.8)
    plt.imshow(binary_image, cmap='Reds', aspect='auto', alpha=0.4)
    plt.colorbar()
    plt.title("Detected Particle Tracks")
    plt.savefig(output_path)
    plt.close()

    with open(csv_path, mode="w", newline="", encoding="utf-8") as file:
        writer = csv.DictWriter(file, fieldnames=["Track", "Intensity",
            "Length", "Angle", "Ionization Rate", "LET"])
        writer.writeheader()
        for i, track in enumerate(tracks):
            writer.writerow({
            "Track": i + 1,
            "Intensity": track["intensity"],
            "Length": track["length"],
            "Angle": f"{track['angle']:.2f}",
            "Ionization Rate": f"{track['ionization_rate']:.2f}",
            "LET": f"{track['LET']:.3e}"
        })
        print(f"Written data to {csv_path}")
```

```
def group_data_by_day(timestamps, seu_counts):
    """
    Group SEU counts by day.
    Args:
        timestamps (list): List of datetime objects.
        seu_counts (list): List of SEU counts.
    Returns:
        tuple: Grouped timestamps and summed SEU counts.
    """
    grouped_seus = defaultdict(int)
    for timestamp, count in zip(timestamps, seu_counts):
        day = timestamp.date()  # Group by date
        grouped_seus[day] += count

    grouped_timestamps = list(grouped_seus.keys())
    grouped_counts = list(grouped_seus.values())
    return grouped_timestamps, grouped_counts

def plot_seus_over_time(timestamps, seu_counts, output_path):
    """
    Plot SEUs over time as a column chart.
    Args:
        timestamps (list): List of datetime objects.
        seu_counts (list): List of SEU counts.
    """
    plt.figure(figsize=(12, 6), dpi=80)
    plt.bar(timestamps, seu_counts, width=0.3, align='center',
        color='skyblue', edgecolor='black')
    plt.xlabel("Time")
    plt.ylabel("Number of SEUs")
    plt.title("SEUs Over Time")
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.xticks(rotation=45)
    plt.savefig(output_path)



if __name__ == "__main__":
    folder_path = 'c:/Users/saras/Desktop/Space Environment/Lab D/full_frame'
    print(f"Using Folder Path {folder_path}")
    if not os.path.exists(folder_path):
        raise FileNotFoundError(f"The provided folder path {folder_path} does
            not exist")
    os.chdir(folder_path) # Change the Execution to folder_path
    threshold = 280 # SEU detection threshold
    marked_images_dir_path = "marked_images"
    os.makedirs(marked_images_dir_path, exist_ok=True)

    data = pd.DataFrame(columns=["Timestamp", "SEU_Count"])


# Variables to track overall statistics
total_heavy_particles = 0
```

```
all_ionization_rates = []

for filename in os.listdir(folder_path):
    if filename.endswith('.bin'):
        csv_index_path = f"track-analysis_results-{filename}.csv"

        # Load and process the full-frame image
        # Load the full-frame image
        full_frame_image = load_full_frame_image(filename)

        # Save the original image as PNG
        original_image_path = os.path.join(marked_images_dir_path,
            f"original_{filename.replace('.bin', '.png')}")
        save_original_image(full_frame_image, original_image_path)

        binary_image = detect_particle_impacts(full_frame_image, threshold)

        # Count SEUs (white pixels in binary image)
        seus = np.sum(binary_image)

        # Extract timestamp
        timestamp = extract_timestamp(filename)

        # Add to dataframe
        data = pd.concat([data, pd.DataFrame({"Timestamp": [timestamp],
            "SEU_Count": [seus]})], ignore_index=True)

        # Analyze tracks
        tracks = analyze_tracks(binary_image, full_frame_image)
        output_image_path = os.path.join(marked_images_dir_path,
            f"{filename.replace('.bin', '.png')}")
        plot_detected_tracks(full_frame_image, binary_image, tracks,
            output_image_path, csv_index_path)

        # Identify heavy particles
        heavy_particles = identify_heavy_particles(tracks, let_threshold=10)
        total_heavy_particles += len(heavy_particles)

        # Collect ionization rates
        all_ionization_rates.extend([track['ionization_rate'] for track in
            tracks])

        # Mark heavy particles
        output_image_path = os.path.join(marked_images_dir_path,
            f"marked_{filename.replace('.bin', '.png')}")
        mark_heavy_particles(full_frame_image, heavy_particles,
            output_image_path)

# Final statistics
if all_ionization_rates:
    ionization_rate_min = min(all_ionization_rates)
    ionization_rate_max = max(all_ionization_rates)
else:
    ionization_rate_min = ionization_rate_max = 0

print("\n--- Summary of Results ---")
```

```python
print(f"Total Heavy Particle Candidates: {total_heavy_particles}")
print(f"Ionization Rate Range: {ionization_rate_min:.2f} SEUs/cm to
    {ionization_rate_max:.2f} SEUs/cm")


# Sort data by time
data["Day"] = data["Timestamp"].dt.date
grouped_data = data.groupby("Day", as_index=False)["SEU_Count"].sum()

# Plot SEUs over time
output_seus_path = os.path.join(marked_images_dir_path, f"seus_count")
plot_seus_over_time(data["Timestamp"], data["SEU_Count"], output_seus_path)
```

Listing 1: Python script for particle track analysis