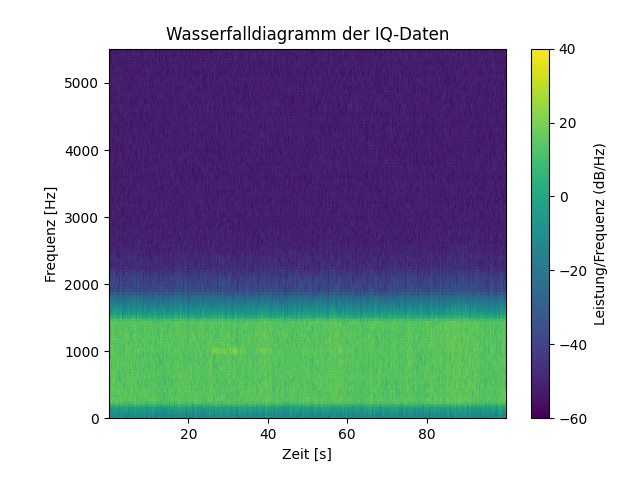
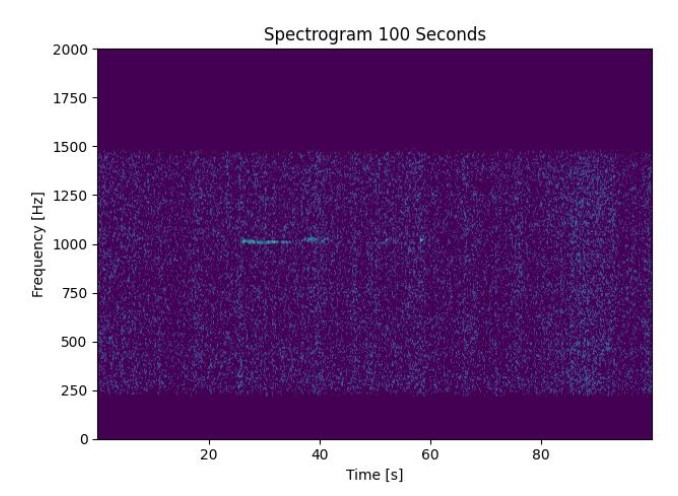
Meteor Scatter Projekt

Von Robert Heinrich und Sebastian Kolb

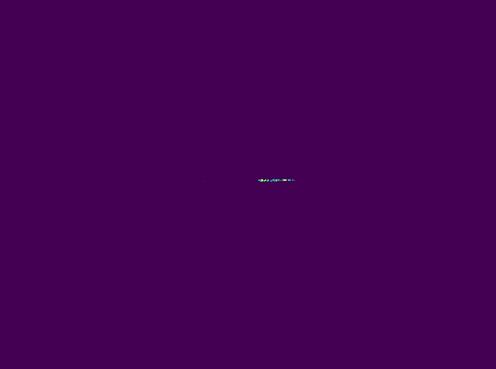
# Spektrogramm Darstellung von Testdaten:

Für den Test wurden die Daten des Astronomie Museums Sonneberg von August 2023 verwendet. Die WAV-Files beinhalten eine Aufzeichnungslänge von 15min. Insgesamt liegen für die Tage 11.-15. August 2023 Daten vor. Die Spektrogramme wurden für einen Zeitbereich von 100s erzeugt. Abbildung 1 zeigt einen ersten Versuch.

Das Signal liegt laut Spektrogramm zwischen 0 und 2000 Hz. Zudem ist ein Burst bei 25 Sekunden und nahe der 60 Sekunden Marke zu erkennen. Um die Bursts besser darstellen zu können wurde nur der relevante Bereich angezeigt und der Leistungsbereich reduziert, um den Burst besser vom umliegenden Rauschen zu unterscheiden.



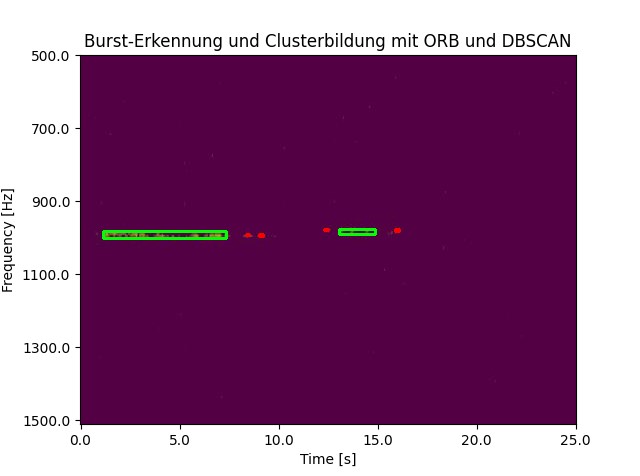
Die Bursts konnten jetzt besser dargestellt werden. Allerdings war das Rauschen immer noch störend. Um ein Spektrogramm frei von Rauschen zu erhalten, wurde die Rauschleistung adaptiv vor der Erzeugung eines Spektrogramms geschätzt. Für die Rauschschätzung wurde der Frequenzbereich 250 Hz bis 800 Hz und damit oberhalb der Signalfrequenz verwendet. Die minimal im Spektrogramm dargestellte Leistung wurde anschließend leicht über der Rauschleistung angesetzt, sodass das Rauschen nicht mehr im Spektrogramm auffällt. Mit dieser Lösung wird auch das Problem schwankender Rauschleistungen berücksichtigt.



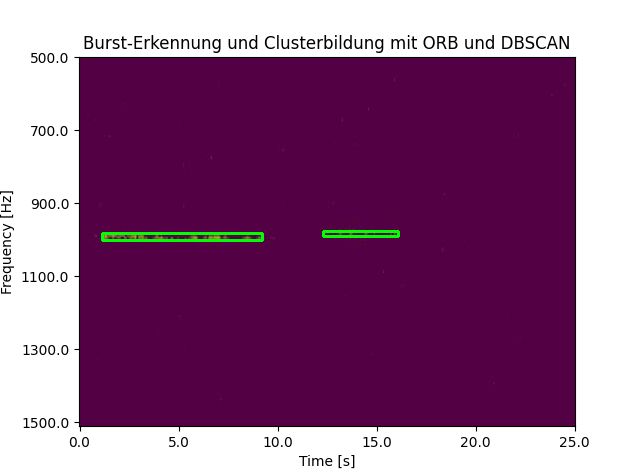
# Burstdetection mithilfe des ORB-Algorithmus

Der ORB-Algorithmus (Oriented FAST and Rotated BRIEF) ist ein Feature-Detektions-Algorithmus, der vor allem in der Computer-Vision eingesetzt wird. ORB verwendet den FAST-Algorithmus (Features from Accelerated Segment Test), um Keypoints in einem Bild zu erkennen. Diese Keypoints sind charakteristische Punkte im Bild, wie Ecken, Kanten oder andere markante Strukturen.

Mit Clustering durch DBSCAN wurden die einzelnen Keypoints zu einem Keypoint-Cluster zusammgefasst. Wenn das Cluster eine bestimmte Anzahl von Keypoints beinhaltet, wird dieses als Burst klassifiziert. Ist die Anzahl der Keypoints jedoch nicht ausreichend, so liegen einzelne starke Rauschpunkte vor die nicht als Meteor-Bursts erkannt werden dürfen. Zudem sollte zwischen überkritischen und unkritischen Bursts unterschieden werden. Überkritische Bursts sind in der anschließenden Darstellung grün dargestellt und unkritische rot. Sie unterscheiden sich in ihrer Länge. So wurden Bursts die länger als 0,5 s sind als überkritisch angesehen und kürzere als unkritisch. Die Zeit pro Spektrogramm wurde zudem auf 25 s und später auf 30 s reduziert.

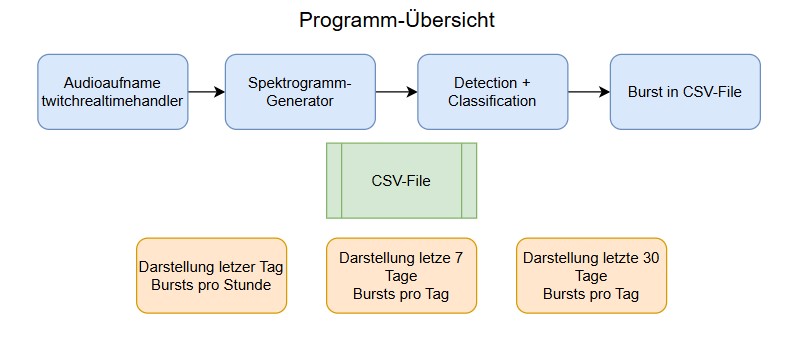


Die Bursts liegen sehr eng beieinander und ein Frequenzversatz ist bei den ersten 3 Clustern nicht zu erkennen. Der Abstand zwischen zwei Clustern auf einen Wert von 5 s eingestellt, sodass ein einzelner Burst nicht als mehrere detektiert wird. Als Resultat erhält man nun 2 grüne also überkritische Bursts:



# Programmaufbau

Das Projekt wurde in zwei Teilprogramme unterteilt. Das Eine ist zur Spektrogramm-Erzeugung und zur Detektion und Klassifikation der Meteor-Bursts verantwortlich. Das Andere für die Darstellung der detektierten Meteore pro Stunde bzw. pro Tag in einem Web-Browser. Als Schnittstelle fungieren CSV-Files. Diese werden von der einen Seite mit Zeitstempeln und der Anzahl der detektierten überkritischen und unkritischen Bursts gefüllt und von der anderen Seite eingelesen. Die folgende Abbildung zeigt eine Programm-Übersicht.



Um die Burst-Detection zu starten muss das Skript prime\_detection.py ausgeführt werden. Dieses Skript greift zudem auf Funktionen aus detector\_and\_classification.py zu.

Die Audioaufnahme wird aus dem twitch-Stream des Astonomie Museum Sonneberg mittels twitchrealtimehandler gewonnen. Sie wird zur Spekrogrammerzeugung kurz zwischengespeichert, um anschließend durch eine Aufnahme der nächsten 30 s ersetzt zu werden. Außerdem verfügt das Skript über die Funktionen load\_wav, list\_audio\_divices und record\_loopback, falls die Audioaufname über ein Loopback Device erfolgen soll. Auf dem RaspberryPi wurde dieser Ansatz getestet. Allerdings ist die visuelle Darstellung des Twitch-Livestreams für den RaspberryPi sehr rechenaufwendig. Durch den Twitchrealtimehandler muss der Livestream nicht visuell dargestellt werden und somit für den RaspberryPi besser geeignet.

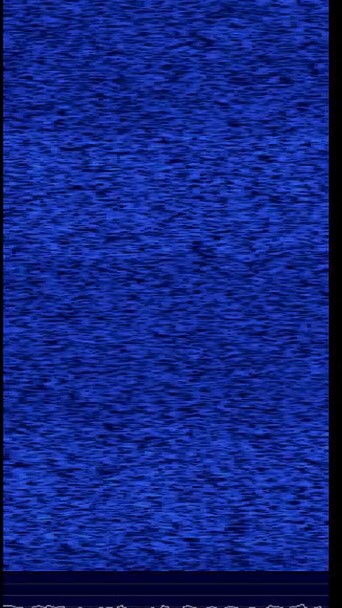
Das Spektrogramm wird in der Funktion plot\_spectrogram erzeugt und das Rauschen mithilfe zuvor erwähnten adaptiven Rauschleistungsschätzung unterdrückt. Es wird als JPG abgespeichert und anschließend vom Spektrogramm der nächsten 30 s überschrieben.

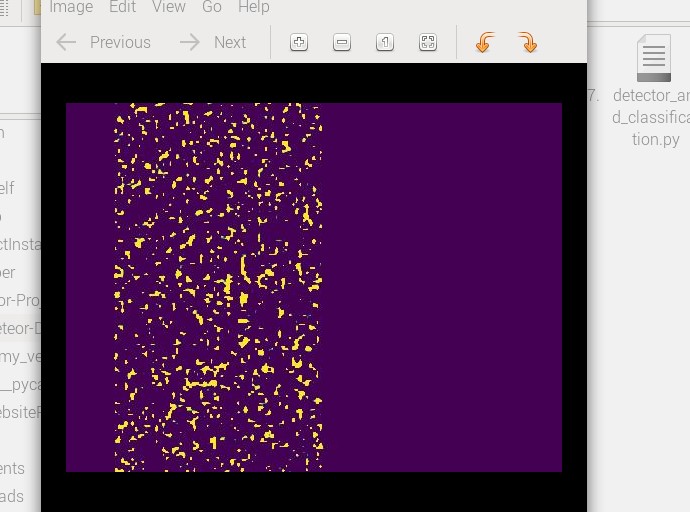
Im erzeugten Spektrogramm erfolgt die Detektion und Klassifikation der Meteor-Bursts. Die Funktion ist im Skript detector\_and\_classificaiton.py zu finden. Das gespeicherte JPG wird eingelesen und durch den ORB-Algorithmus ausgewertet. Die Keypoints werden in einem Array hinterlegt. DBSCAN bildet daraus Keypoint-Cluster. Diese besitzen die Eigenschaften Anzahl von Keypoints und maximalen Abstand Epsilon benachbarter Keypoints. In mehreren Performance-Tests, bei denen die Detektionsrate von zuvor gelableten Testaufnahmen mit verschiedenen Werten für min\_samples (Anzahl Keypoints) und eps (Epsilon) bestimmt wurde, haben sich min\_samples = 5 und eps = 30 als zuverlässige Parameter ergeben. Schließlich werden die Cluster anhand ihrer Länge in kritische und unkritische Bursts kategorisiert. Hierbei werden Bursts die über 500ms anliegen als kritisch und Bursts die kürzer anliegen als unkritisch eingestuft.

In einer Schleife werden die Punkte Audioaufnahme, Spektrogramm-Erzeugung, Burst Detektion und Kategorisierung alle 30s wiederholt. Durch zwei zeitliche Bedingungen in der Schleife wird überprüft ob eine Stunde oder ein ganzer Tag vergangen ist. Jede Stunde wird ein Eintrag in einem CSV-File vorgenommen. Dieser enthält einen Zeitstempel, die Anzahl der detektierten Bursts und die Anzahl der kritischen Bursts. Ist ein ganzer Tag vergangen, so wird ein neues CSV-File angelegt und ein Eintrag in diesem erstellt. Die CSV-Files sind nach dem Datum benannt, um Nameskonflikte zu vermeiden (z.B. 20250109.csv für den 09.01.2025).

Es kann vorkommen das die Audioquelle, der Twitchlivestream, für einige Sekunden aussetzt. In diesem Fall wäre das Spektrogramm nicht vollständig und die Burst-Detektion wäre fehlerhaft. Eine Bedingung in der Schleife überprüft die Array-Länge der Audioaufnahme. Diese sollte immer 150000 (5000 1/s Samplerate \* 30 s) betragen. Ist diese kleiner so wird die Audioaufnahme verworfen. Für diesen Zeitraum können keine Bursts detektiert werden. Das Programm startet anschließend eine neue Aufnahme und wird wie zuvor fortgesetzt.

Eine Fehlerquelle kann allerdings nicht vom Programm selbst aufgefangen werden. Es ist schon häufiger vorgekommen das sich der Empfänger im Astronomie Museum aufhängt. Die folgende Abbildung zeigt das Spektrogramm des Twitch-Livestreams in dem hellere und dunklere Bereiche durch den Empfänger falsch erzeugt werden. Wenn der Empfänger neugestartet wird, tritt dieser Fehler für die nächste Zeit nicht mehr auf. Eventuell wäre es notwendig einen täglichen automatischen Neustart um Nachmittag (geringste Meteoraktivität) vorzunehmen. Sonst werden sehr viele falsche positive Bursts detektiert, die mit unserem Ansatz nicht von richtigen Bursts unterschieden werden können. Stand 13.01.2025 Screenshots sind vom 12.01.2025.





Der Web-Browser wird über app.py gestartet und kann anschließend über den localhost und Port 5000 aufgerufen werden.

127.0.0.1:5000