Hardware accelerated onboard image processing for space-based meteor observation: Concept and Implementation of SpaceMEDAL

Jona Petri[(1)], **Julia Zink**[(1)], **Sabine Klinkner** [(1)]

[(1)] *IRS, University of Stuttgart, Pfaffenwaldring 29, D-70569 Stuttgart, +49-711-685-63094, petri@irs.uni-stuttgart.de;klinkner@irs.uni-stuttgart.de*

ABSTRACT

The University of Stuttgart's Institute of Space System is currently developing the CubeSat Stuttgart Operated University Research Cubesat for Evaluation and Education (SOURCE) in cooperation with Small Satellite Student Society at the University of Stuttgart (KSat e.V.). The mission of the three-Unit CubeSat is dedicated to technology demonstration and demise investigation using a sensor suite for atmospheric and re-entry science. Furthermore, the satellite is equipped with a meteor observation camera. During an eclipse, this camera is pointed towards Earth and is continuously taking images in order to observe a meteor. The main advantages of a space-based meteor observation are the greater coverage as well as weather independence. However, the downlink capacity of a small satellite, especially a CubeSat, is limited. Thus, the satellite uses onboard processing of the images to detect meteors. The onboard meteor detection algorithm called *Spaceborne MEteor Detection ALgorithm (SpaceMEDAL)* faces two main challenges: First, *SpaceMEDAL* must detect faint, short, and fast-moving events while the background is moving due to the satellite motion. This requires the usage of a new detection method. *SpaceMEDAL* is based on the optical flow calculation, a method in image processing that gives the movement of each pixel between two frames. Here, the second challenge arises: The optical flow calculations are computationally intensive, but the processing resources on a CubeSat are limited. This paper presents the novel onboard meteor detection algorithm *SpaceMEDAL* based on optical flow calculations and its implementation for the CubeSat SOURCE. Both challenges described above are addressed. First, the concept of the algorithm, the optimization of its parameters, and detection performance are given. The algorithm is based on optical flow calculations, background determination, and object detection using the *OpenCV* library. The detection performance is evaluated by a dedicated simulation to generate test data and hardware-in-the-loop Testbed. It can be shown, that 78.8 % of all meteors in the test set can be detected with a low rate of false positive detections (5.3 %). Second, the implementation on the *SOURCE* payload computer is given. In order to allow the processing of all images in a reasonable time with the limited hardware resources, the Field Programmable Gate Array (FPGA) of the Payload On-Board Computer (PLOC) is used. The most resource intensive function of the algorithm, the optical flow calculation, is exported to the FPGA. This accelerates the function by a factor of 5.2 and the algorithm by a factor of 3. This is possible by using tools provided by the FPGA manufacturer. All in all, the presented detection algorithm and its implementation are a crucial part of the SOURCE mission and potential successors. Furthermore, the implementation of *SpaceMEDAL* using hardware acceleration have a high potential to be adapted to other missions requiring onboard processing.

Acronyms

API Application Programming Interface; **ArtMESS** Artificial Meteorvideo Simulation Software; **CPU** Central Processing Unit; **CVB** Common Vision Blox; **FN** False Negative; **FP** False Positive; **FPGA** Field Programmable Gate Array; **FSFW** Flight Software Framework; **IRS** Institute of Space Systems; **ISS** International Space Station; **KSat e.V.** Small Satellite Student Society at the University of Stuttgart; **MeSHCam** Meteor Observation, Star and Horizon Tracking Camera; **OBC** Onboard computer; **PCDU** Power Control and Distribution Unit; **PERC** Planetary Exploration Research Center; **PLOC** Payload On-Board Computer; **PRIma** PR Imager; **PUS** Packet Utilisation Standard; **SOURCE** Stuttgart Operated University Research Cubesat for Evaluation and Education; **SpaceMEDAL** Spaceborne MEteor Detection ALgorithm; **TP** True Positive

1 INTRODUCTION

The Institute of Space Systems (IRS) at the University of Stuttgart is currently developing the CubeSat *Stuttgart Operated University Research Cubesat for Evaluation and Education (SOURCE)* in cooperation with Small Satellite Student Society at the University of Stuttgart (KSat e.V.). The mission of the three-Unit CubeSat (see Figure 1) is dedicated to technology demonstration and demise investigation using a sensor suite for atmospheric and re-entry science (see [1]). Currently, the project is in Phase D, the satellite is being built and the launch is planned for 2023, see Table 1 for more mission details.

Table 1. *SOURCE* mission facts

Property	Value
Orbit	Sun-synchronous/ISS orbit
Initial orbit altitude	450 km to 500 km
Mass	$\sim$ 5 kg
Size	30 cm $\times$ 10 cm $\times$ 10 cm
Mission duration	$\sim$ 1 year
Attitude determination	Sun sensors, magnetometer, experimental Startracker
Attitude control	Magnetorquer
Pointing accuracy	5°
Power generation	10 solar panels (327 $\times$ 80.25 $\times$ 1 mm) á 7 cells
Maximum power generation	30 W
Payload data rate per day	100 MB using S-band

The satellite is equipped with two cameras: One camera is called PR Imager (PRIma) and used to take images for public relation. The second camera is dedicated to meteor observation. This camera is called Meteor Observation, Star and Horizon Tracking Camera (MeSHCam) (see Figure 2) and consists of an industrial machine vision camera (*GenieNano M1920*) equipped with a *Schneider Kreuznach Cinegon 1.4/12* lens. During eclipse ($\sim$ 30 min per $\sim$ 90 min orbit), the camera is pointing towards Earth and continuously taking images with 6 *fps* to 10 *fps*. Meteors are light phenomena whose exact time and place of occurrence are random.

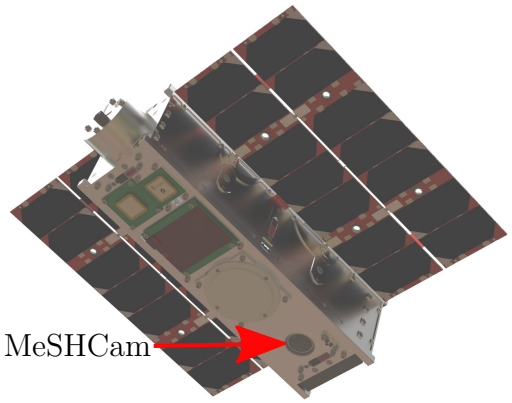


Figure 1. Rendering of the *SOURCE* Satellite. MeSHCam is located on the right side.

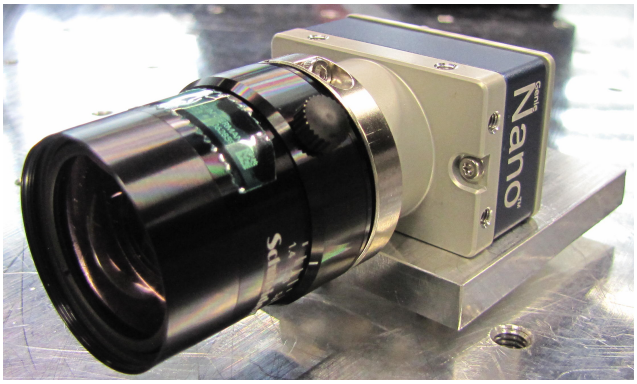


Figure 2. Image of MeSHCam mounted on a plate for tests in the optical laboratory.

Thus, continuous observation is required. Since this observation concept generates a large amount of data, which can not be downlinked given the limited capabilities of a CubeSat, the images need to be processed onboard. Thus, a meteor detection algorithm is needed to detect meteors in the images and select only those images for downlink.
The algorithm faces two main challenges: First, resources (like power) on a CubeSat are limited. This limits the available processing power of the Payload On-Board Computer (PLOC) and the time the algorithm can run due to limited electrical power. Thus, an efficient and fast implementation is required. Second, the algorithm faces intrinsic challenges associated with meteor observation in general and with space-based observation in particular: Meteors are a fast moving (several $\mathrm{km\,s^{-1}}$), usually faint and short (2 s to 4 s) light phenomena. This makes them challenging to detect. Nevertheless, several ground-based algorithms exist and are used for ground-based observations (see [2]). However, these algorithms are not applicable for a space-based algorithm, since they require a static background. Because the satellite is moving during observation, the background is moving as well and methods based on a static background do not work. Thus, we developed a new algorithm for onboard meteor detection called *Spaceborne MEteor Detection ALgorithm (SpaceMEDAL)*.
In this paper we describe three main parts of the *SpaceMEDAL* development: First, the concept of our novel algorithm (see Section 2) is outlined. Next, our approach to testing and optimizing the algorithm (see Section 3) is presented. This includes data generation for testing, a hardware-in-the-loop test setup and automatic data evaluation, and algorithm parameter optimization. Finally, the implementation and acceleration of the algorithm using the Field Programmable Gate Array (FPGA) of the PLOC is presented (see Section 4).

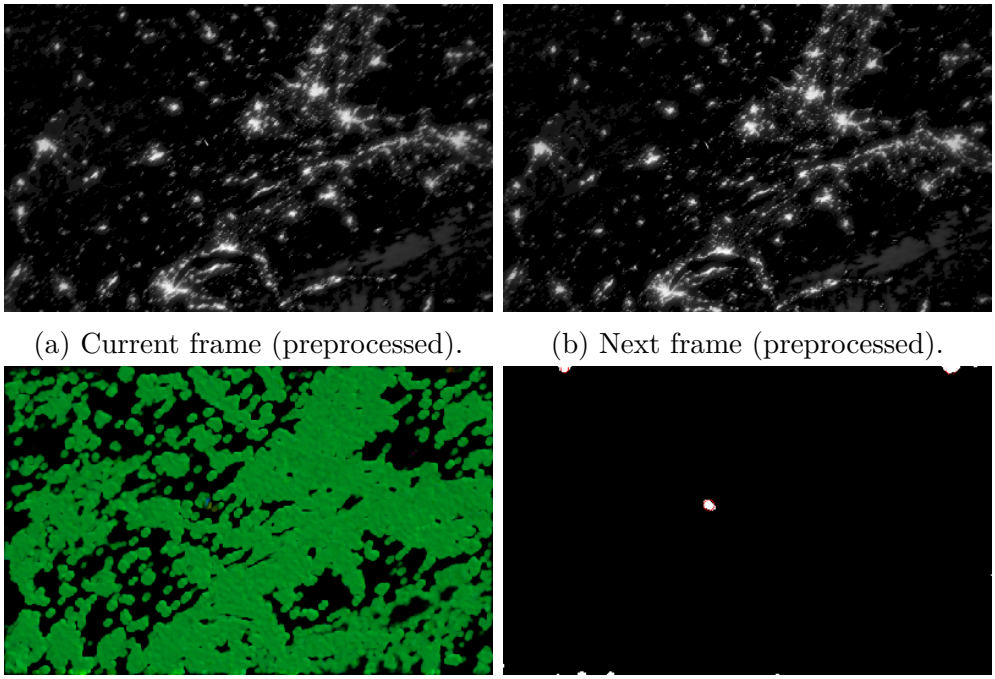
2 SpaceMEDAL ALGORITHM CONCEPT

The basic idea followed by *SpaceMEDAL* is that meteors usually move differently than the background. Hence, the algorithm determines the movement of the whole image scene and detects features that move at different speeds and directions than the overall scene. The movement of the image is calculated using Farnebäck's dense optical flow algorithm (see [3]). Features of interest are extracted by blob detection. If similarly moving blobs are detected in a

certain amount of frames, these blobs get classified as a meteor. The algorithm is implemented in *C++* and uses image processing methods of the *OpenCV* library.

2.1 DETAILED ALGORITHM CONCEPT

The algorithm starts with reading input parameter values from an external configuration file, the `config.cfg` file. These input parameters set parameters of *SpaceMEDAL*, e. g. to define the properties a detected feature has to show to be classified as a meteor. Furthermore, they define the settings for preprocessing and optical flow calculations.
Afterwards, the algorithm proceeds to consecutively read in images taken by *MeSHCam*. Thereby, two subsequent frames are examined for meteor detection. Both images are preprocessed by thresholding dark pixels. This means that all pixels below the threshold value are set to 0. Thresholding removes noise, hides out irrelevant regions, and increases the contrast in the images. Tests showed that preprocessing is an important step to obtain a clear image for blob detection.
After preprocessing, the dense optical flow between the two frames is determined. The optical flow gives the displacement in the X and Y direction of each pixel between frames. On basis of the calculated displacement vectors of each pixel, an angle and magnitude image are derived. These describe the magnitude and direction of motion (angle) of the whole scene respectively. The algorithm then calculates the mean and standard deviation of angle and magnitude and uses these values to determine the main motion of the background.
The next step is to acquire an image that is suitable for blob detection. To do so, the main motion of the background is masked out in the magnitude of motion image, and the remaining noise is removed by a median filter. The resulting image shows the magnitude of motion of non-background regions between two frames.
Finally, the resulting image is converted to a binary image by setting all pixels with a value above 0 to 255. This highlights all non-background pixels and facilitates blob detection. In this binary image, the algorithm detects blobs by applying *OpenCV* 's `cv::SimpleBlobDetector`. Figure 3 shows the image processing steps conducted by the algorithm.
In order to distinguish between blobs caused by elements in the background (e. g. lightning) and blobs which could possibly be meteors, blobs are filtered by their area and circularity. The typical area and circularity of blobs caused by meteors could be determined experimentally. Additionally, the algorithm compares the standard deviation of a blobs intensity to a threshold value (defined by input parameters). A blobs intensity is determined in the respective preprocessed current frame. If the detected blobs standard deviation is below the threshold, the blob most likely belongs to the background and is discarded. The remaining blobs are considered potential meteors and temporarily stored.
Assuming that a meteor has the same movement across multiple frames, each potential meteor's movement (direction and speed) is compared to previously detected blobs within a specific frame range. As soon as the algorithm finds similarly moving blobs in a certain amount of frames, the detected blob is classified as a meteor. Its properties, such as frame number, blob center coordinates, and size as well as angle and magnitude are exported to a `.csv` file.
In the next step, the detection performance of *SpaceMEDAL* must be evaluated and the parameters must be adapted. Thus, in the next section, the generation of test data as well as the method of parameter optimization is outlined.



(a) Current frame (preprocessed). (b) Next frame (preprocessed).

(c) Optical flow between frames (d) Image for blob detection.

Figure 3. Visualisation of image processing steps of meteor detection algorithm. The white dot (blob) in the middle of image (d) represents a successful detection of a meteor.

3 SpaceMEDAL ALGORITHM OPTIMIZATION

A crucial step in the *SpaceMEDAL* development is a proper testing approach, to ensure the algorithm is working as intended. Furthermore, different algorithm parameters need to be set and their effect evaluated depending on the imaged scene. This requires a suitable set of test data. Thus, a Python script to generate a systematic test set was developed (see Section 3.1). Additionally, a setup in the optical laboratory of IRS was built, to allow testing using the actual hardware in a realistic environment (see Section 3.2). The data generation script, as well as the test setup, are described in another paper (see [4]), here only an overview is given.
After generating the required test data, we optimized the input parameter values on a data set covering various meteor events. The optimization allowed us to find a parameter setting that delivers a high and reliable detection rate for different meteor events. We developed a Python script that automates the process of testing and parameter optimization (see Section 3.3).

3.1 Testdata generation

The only video data showing a meteor from orbit is taken from the International Space Station (ISS) by the *METEOR* project from Planetary Exploration Research Center (PERC) (see [5]). Using these videos for the *SpaceMEDAL* development is possible, but does not allow for systematic testing of different meteor properties. Furthermore, the camera properties and settings (especially the frame rate) influence the representation of a meteor in the image and therefore the design of the detection algorithm. Also, the videos are only available in a low resolution.

Thus, the Python software *Artificial Meteorvideo Simulation Software (ArtMESS)* was developed. This software allows to generate a wide range of test data required to test the algorithm. Since *SpaceMEDAL* must be able to detect meteors with different properties (velocity, brightness, and duration), multiple videos with different meteor properties are generated to systematically test the algorithm. The test set is also used to evaluate the effect of different algorithm parameters on meteor detection by running the algorithm with different parameters on the same dataset and evaluating the effect on the detection rate.
The test set consists of videos showing a meteor from the perspective of a satellite (see Figure 4). Therefore, images containing a meteor are generated depending on various settings, which are combined with different available backgrounds. This is done by overlaying the background image with the meteor image, storing the combined images as a video frame, and finally upscaling the resulting video to increase resolution.

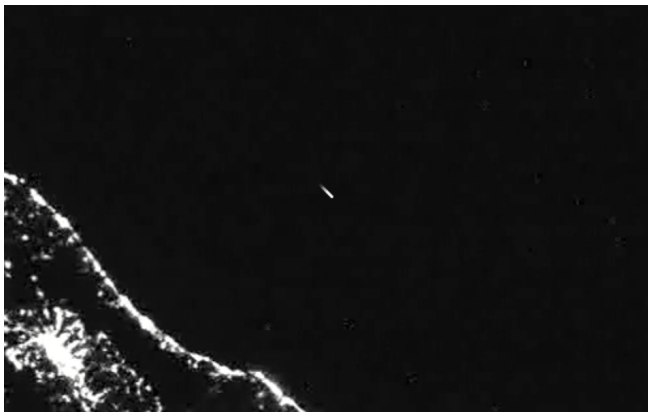


Figure 4. An artificially generated meteor implemented into a video taken from the ISS by the PERC project [a]

[a] `https://www.youtube.com/watch?v=2DVsYWhvf0U`

Three different types of backgrounds are used for different purposes: A black background (only the meteor is shown) is used to evaluate the effect of different meteor properties on the algorithm performance. Furthermore, different *SpaceMEDAL* parameters and their effect can be tested. Next, the black marble images[1][2] showing the Earth at night are used to generate a moving background including city lights. This allows to test the algorithm with a more realistic background and evaluate the effect of city lights. Additionally, it is possible to implement a rotation around the optical axis. This simulates the rotation of the satellite in case the attitude control system is not able to stabilize the satellite perfectly. Finally, the videos taken from the ISS by the PERC project are used to generate videos with a realistic background including city lights, clouds, and lightning.
Since the *ArtMESS* tool allows to generate a lot of videos fast, it is used to generate different test sets with different meteor and background properties allowing to optimize the algorithm parameter set for specific videos. The *ArtMESS* tool also exports the position and frame number of each implemented meteor, to allow an automatic evaluation after the algorithm processed the videos.

[1] `https://blackmarble.gsfc.nasa.gov/`
[2] `https://earthobservatory.nasa.gov/features/NightLights`

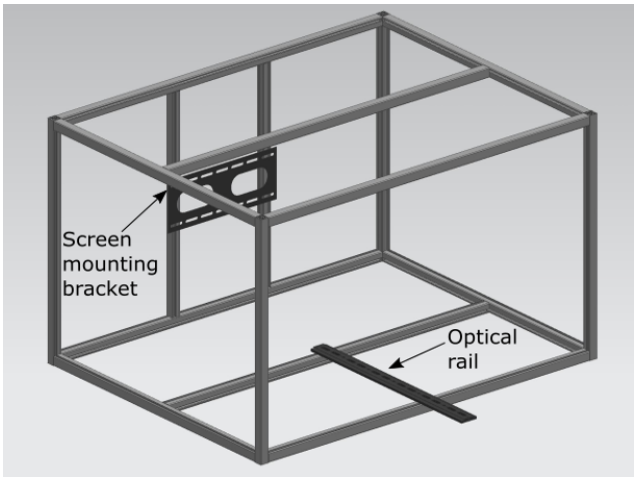


Figure 5. CAD Rendering of the Testbed frame, showing also the optical rail for camera mount and bracket for screen mount.

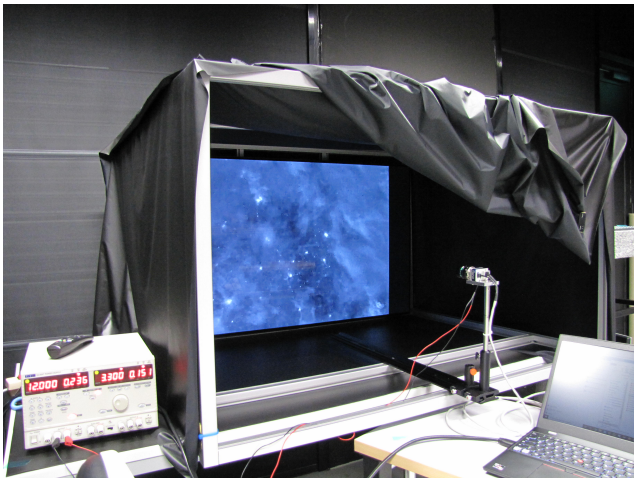


Figure 6. Image of the Testbed during algorithm testing. A meteor video is displayed, usually the curtain is closed during testing.

3.2 Testbed setup for realistic data generation

The videos can be directly processed by the algorithm, after the test set was generated. However, camera settings (exposure time) and noise influence the algorithm. Thus, a Testbed was developed to allow for realistic testing. The Testbed consists of a large OLED screen mounted on an aluminium frame. The camera is mounted on an optical rail to allow horizontal and vertical adjustments. An image is shown in Figure 5 and Figure 6. When conducting measurements, the complete setup is covered in a blackout curtain to prevent stray light from entering the setup.

Furthermore, a small Python script is used to semi-automatically image multiple videos: This script generates folder names from the videos which should be displayed, controls the camera, and stores all images in the according folder. The operator only has to start imaging and video display at the same time. This allows to capture a large amount of image data with low effort, thus allowing to test the algorithm with a lot of data. More details about the *ArtMESS* tool and the Testbed calibration can be found in [4].

3.3 Automatic data evaluation and parameter optimization

After imaging all videos, the test data must be processed by *SpaceMEDAL* and the results evaluated.

Our goal is to find a combination of input parameters that achieves a high detection performance for a variety of meteor events. The *SpaceMEDAL* parameters influence the preprocessing of the images, the calculation of the dense optical flow, and the calculation of the main motion of the background. Further, the input parameters define the properties a blob has to fulfil to be classified as a meteor. These properties amongst others include the area and circularity of a blob or the standard deviation of a blobs intensity.

In order to evaluate the performance of *SpaceMEDAL*, we consult the precision and the recall values, two common metrics to evaluate object-detection algorithms. Precision P is the proportion of correct detections and describes the algorithm's ability to identify only relevant objects.

Recall R is the ratio of correct detections to all ground truths and describes its ability to find all relevant cases. The two values are calculated as follows (see [6]):

$$P = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}} \tag{1}$$

$$R = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}} \tag{2}$$

The harmonic mean of precision and recall is called *F-measure* F (see [7]):

$$F = \frac{2 \cdot P \cdot R}{P + R} \tag{3}$$

For *SpaceMEDAL*, we define a True Positive (TP) as a meteor that is detected in at least one frame by the algorithm. Every false detection is a False Positive (FP) and every meteor which is not detected at all is a False Negative (FN).
We developed a data evaluation script in Python which automates the process of running *SpaceMEDAL* with different test data and parameters. Further, the script evaluates the respective detection performance.
The script first determines the expected meteor positions in the provided camera images (ground truths). This is done using the exported meteor positions from *ArtMESS* and the geometric Testbed calibration linking screen to camera pixel positions. Then, the script adjusts the values of the input parameters in the `config.cfg` file and starts the meteor detection algorithm. After *SpaceMEDAL* processed all images of the data set and exported the positions of detected meteors, the Python script compares these detections to the ground truths, identifies TPs, FPs and FNs and calculates recall, precision, and *F-measure*. A detailed explanation of the script can be found in [8].
We first manually adjusted the input parameter values. This helped us to understand the effects and relevance of different parameter values and meteor properties on the detection performance. Further, we were able to identify a favourable range of values for each parameter. Afterwards, we used these parameter values as an input for a Nelder-Mead optimization (see [9]). The Nelder-Mead optimization is based on the comparison of function values and does not require a function gradient. Thus, it is suitable to optimize the input parameters of our meteor detection algorithm.
The optimization is conducted with Python's `scipy.optimize.minimize()` function (see [10]). This method takes an objective function of one or more variables and minimizes its return value. The Nelder-Mead optimization approaches the sought minimum by iteratively adjusting the values of the variables depending on the return value of the objective function.
For *SpaceMEDAL*, the goal is to achieve a high detection performance which is indicated by a high *F-measure*. Hence, the optimization minimizes the negative *F-measure*. The objective function is a method that creates the `config.cfg` file containing the input parameter values, executes the meteor detection algorithm, and evaluates its performance based on the automatic data evaluation script. It returns the negative *F-measure*.
SpaceMEDAL uses a total of 30 parameters. In order to facilitate the optimization, only the six parameters which we identified to have the largest impact on the meteor detection were optimized. The selected parameters mainly influence which types of blobs are considered a meteor and how the background motion is determined. We selected 113 videos (black marble and ISS background) featuring meteors of different angles, brightness, and speed for the optimization.

With the optimized parameter values, the meteor detection algorithm detected 89 of 113 meteors while only detecting 5 FPs. Hence, 94.68 % of all detections are correct ($P = 0.9468$) while 78.76 % ($R = 0.7876$) of all meteors were detected. The high *F-measure* of 0.8599 indicates a very robust and reliable meteor detection algorithm.
Although the optimized parameter values deliver great performance for the consulted test data, the parameter values and their respective performance should be interpreted as references and not as specifications. This is because only a limited amount of data was used for the optimization. Hence, other parameter values might be more suitable for different meteor events and in-orbit optimization of the parameters might be necessary. Further, due to the short duration of the videos, the frequency of meteors in the test data is higher than expected in orbit. It is possible that the number of false detection increases in orbit, where the frequency of meteors is lower.

3.4 SUMMARY

The presented space-based meteor detection algorithm works well and is able to detect meteors of different properties, such as velocity and brightness, in front of different backgrounds. Our novel approach of using a combination of optical flow calculations and blob detection to determine parts of the image moving different than the background is a promising solution for the challenge of space-based meteor detection.
Additionally, during the algorithm development, several software tools and procedures have been developed, which can also be used for other space-based meteor detection missions. These tools are a crucial part of the development effort since they allow for fast, low-effort, and realistic test campaigns. The tools include the software to generate test data (*ArtMESS*), the Testbed setup and calibration as well as the automated evaluation of detection performance and optimization of the algorithm's parameter. All these tools can be easily adapted for different instruments.

4 SpaceMEDAL ALGORITHM IMPLEMENTATION

As shown in the previous section, *SpaceMEDAL* delivers a good detection performance. However, for a successful mission, the algorithm must process the images on the PLOC in a reasonable time. The time needed to process one image has a huge influence on the scientific output of the mission. The longer the processing needs, the more power is used and thus fewer images can be taken and processed, decreasing the chance of successful meteor observation (see [11] for more details on the analysis of the scientific output). Therefore, the efficient implementation of the algorithm is crucial for mission success.
Furthermore, the algorithm must be controlled remotely, including configuring parameters. Thus, the algorithm control must be implemented in the PLOC software. The PLOC software is based on the *Flight Software Framework (FSFW)* developed at the IRS, which allows remote commanding and control of a satellite (see [12]).
In this section, the used hardware for the PLOC, the software tools used to develop a hardware-accelerated algorithm as well as details about the algorithm implementation in the PLOC software is given.

4.1 HARDWARE

The *SOURCE* PLOC (see Table 2) is based on the Trenz TE0720-03-1CFA. The board is mounted on the so called port expander, a circuit board designed by the *SOURCE* team. The port expander connects different components of the satellite bus to the Onboard computer (OBC). For the payload subsystem, the board is mounting the PLOC board, implementing interfaces to the *MeSHCam* and *PRIma* as well as to the OBC to receive commands. Furthermore, the SD Card used for image storage and the PLOC software is located on the port expander. The *MeSHCam* and *PRIma* are directly connected and controlled by the PLOC. However, the power supply of both cameras is controlled by the Power Control and Distribution Unit (PCDU).

Table 2. Data sheet *SOURCE* PLOC

Property	Value
Mass memory (SD Card)	32 GB
Working memory	1 GB
Processing unit	ARM dual-core Cortex-A9 MPCore
Interfaces	Gigabit Ethernet, USB 2.0, I2C, SPI
Power consumption	$\sim$ 3.5 W

4.2 SOFTWARE

The general software outline is shown in Figure 7. The operating system used is Linux, which runs the Autostart software as a system service, a small software which starts the PLOC software. PLOC software is the term of the main software used for camera control and image processing. The PLOC software is using the *FSFW* as well as Common Vision Blox (CVB) for camera control and *OpenCV* for image processing.

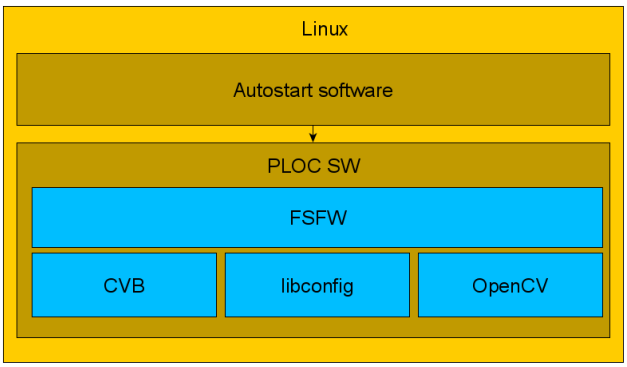


Figure 7. Overview of *SOURCE* instrument software design

Operating System The operating system used on the PLOC is the Linux distribution Ubuntu 18.04 LTS, which allows to make use of the standard software repository to install necessary packages (e. g. libraries needed for the *FSFW* or *OpenCV*). Since the operating system is running on an embedded board, it is necessary to build a custom Linux kernel as well as the boot loader responsible for starting the system.

The custom kernel is generated using the software tools from the manufacturer called Vivado and Peta Linux. While Vivado is used to configure the hardware (e. g. interfaces and processor clock frequency), Peta Linux is used to configure and generate the Linux kernel based on the configured hardware. The configuration of the operating system after successfully booting the system includes installing all necessary dependencies as well as needed software (e. g. *OpenCV* and CVB). Also, the Autostart software, which is used to first establish communication with the OBC and start the PLOC software, is configured to start automatically after booting as a system service.
With these steps the configuration of the PLOC and the operating system basis is finished and able to run the PLOC software. More information about the individual parts of the PLOC software is given in the following.

Flight Software Framework As a basis for the PLOC software (written in *C++*) the Flight Software Framework (FSFW) developed at the IRS is used (see [12] for more details). The usage of *FSFW* allows for a faster software development since the framework provides functions and interfaces required for a satellite software. The framework includes functions to generate independent tasks for every object, functions to communicate between those task as well as functions to process and distribute Packet Utilisation Standard (PUS) messages. PUS messages are used in satellite communications to command and control a satellite. In the *SOURCE* satellite the PLOC is commanded by the OBC, which is commanded from ground using commands based on the PUS. Therefore, an easy way to command the PLOC is to forward packets from the OBC. Thus, using the *FSFW* with the already implemented PUS standard allows for an faster development. Furthermore, the development effort could be reduced, by using similar code implemented in the *SOURCE* OBC, e. g. reading the serial communicating port which connects PLOC and OBC as well as the code to control the file system.
Since the framework is an object orientated framework written in *C++*, all needed functionalities are implemented in objects. For the PLOC software, the most important objects for the algorithm implementation are:

- *MeSHCam* Handler
- *SpaceMEDAL* Handler
- File System Handler
- Serial Interface Handler
- System Handler

The *MeSHCam* handler is responsible for controlling the camera (set parameters, take images), conducting meteor observation and store meta data for each observation (number of images taken, image names). This information is used by *SpaceMEDAL*, to process the according images and keep only the images containing a meteor. The file system handler is responsible for data exchange with the OBC (e. g. software updates and images), deleting images and providing information on the file system (e. g. number of stored images, free storage space). The system handler provides functions to control the operating system, e. g. execute commands in the command line, compile software and apply software updates. Messages are received via the serial interface, which is controlled by the serial interface handler.
Each object implements functionalities from the *FSFW*, mostly interfaces (`IF`) used to communicate with the object and execute functions. Those interfaces include executing commands

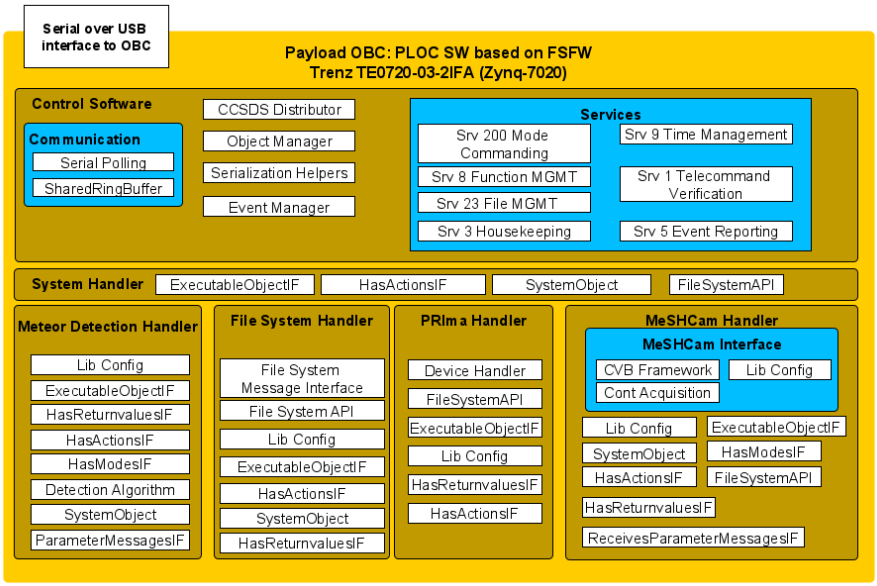


Figure 8. Detailed overview of *SOURCE* instrument software design

(`HasActionsIF`), providing different modes (`HasModesIF`) and receive parameters and messages (`ParameterMessagIF`).
A more detailed layout of the PLOC software is shown in Figure 8.

libconfig libconfig is a library used to read configuration files in *C++*. A configuration file contains values for variables, which are used in the software. By changing those values, it is possible to change the behaviour of the program without having to recompile it. In the PLOC software, configuration files are used by the *MeSHCam* and *SpaceMEDAL* handler to store camera and algorithm parameters.

OpenCV *OpenCV* is an open-source software library, used to process the images with *SpaceMEDAL*. For *SOURCE* it us used in version 3.4.0, since this version is used by the hardware acceleration (see Section 4.4). The main functions used are the optical flow and blob detection functions as well as some functions for image processing (e. g. thresholding). For more details, see the algorithm section (Section 2).

CVB Common Vision Blox is a framework from *Stemmer Imaging* to develop applications for cameras using the *GigE Vision* interface. It offers different Application Programming Interfaces (APIs), to control the camera and take images. The *C++* interfaces are integrated in the *MeSHCam* handler object.

4.3 COMBINATION of FSFW and ALGORITHM

The algorithm is not directly implemented into the PLOC software, instead *SpaceMEDAL* is a separate binary file called using the `CommandExecutor` of the *FSFW*. This solution is chosen due to three main reasons: First, the PLOC software and *SpaceMEDAL* are both complex software, but with different tasks and thus design decisions. Merging both software would require major redesigns in the algorithm and violate design principles of the *FSFW*.

Second, the algorithm uses hardware acceleration as described in the next sections. This requires a separate special compiler to compile the binary file of the algorithm. Using this compiler with the *FSFW* would require changes in the complex build system.
Third, using a separate binary makes updating the algorithm easier, since the software can be compiled on ground and send to the PLOC without changing the PLOC software, which increases reliability.
The `CommandExecutor` allows to execute the algorithm in non-blocking mode, meaning the PLOC software can continue running, while the algorithm is executed. The output of the algorithm is passed to the PLOC software, thus the status of the algorithm can be monitored.
The algorithm reads the configuration file after being started. Here, the folder containing the images which should be processed is specified. The images from each observation are stored in a folder called *OBS_X* with *X* being the number of the current observation. Furthermore, each image of the observation is also numbered consecutively. In the configuration file, the last number of the *OBS* folder processed as well as the last image processed is stored. Thus, the processing can be continued after stopping the algorithm or a specific folder to be processed can be set from ground by modifying the configuration file.
If a meteor is detected, the according images are stored in a *PROC_X* folder, which can be downlinked.
The algorithm is stopped either automatically once all images are processed or using a stop command. The stop command is either issued by the OBC or by the PLOC software if the algorithm is set to run for a specific amount of time.

4.4 HARDWARE ACCELERATION

As mentioned, besides the detection performance, the speed performance is a crucial aspect of the algorithm. It describes how many frames the algorithm processes per second. It is crucial to increase the speed performance, in order to increase the scientific output of the mission.
Since the *SOURCE* PLOC has a FPGA, it is possible to increase the processing speed, by outsourcing parts of *SpaceMEDAL* into the FPGA. This means, some functions are implemented directly in the hardware of the FPGA and thus are not executed by the Central Processing Unit (CPU). A function implemented in hardware works usually orders of magnitude faster.
The manufacturer of the FPGA offers the possibility to compile a program with some functions of the program implemented in hardware. Several steps and multiple software packages are required before the hardware acceleration is working, in this paper only the results are given.
The actual hardware acceleration is implemented in a software called *SDSoC*. This software is needed to compile the algorithm with selected functions exported to hardware.
The existing algorithm source code (written in *C++*) is imported into *SDSoC*. Since the image processing functions (optical flow and blob detection) from *OpenCV* are the most time-consuming, those are candidates for hardware acceleration. In a first step, the optical flow calculations are exported into hardware. The function call `calc-Optical-Flow-Farneback` is replaced by an identical function from *xfOpenCV*[3]. The *xfOpenCV* library provides kernels containing *OpenCV* functions optimized for the used FPGA. After replacing and selecting the according function for hardware acceleration, the algorithm is compiled and the binary file can be executed on the PLOC.

[3]See `https://github.com/Xilinx/xfopencv`

For comparison, the same algorithm without hardware acceleration is also run on the PLOC. As an example, one of the test set videos is processed with both versions and the time needed for different processing steps is measured. The results are shown in Table 3. As can be seen,

Table 3. Final algorithm speed performance. The processing time for one frame is shown. The optical flow step includes the optical flow calculations as well as background subtraction.

Processing step	Without acceleration (ms)	With acceleration (ms)
Optical flow calculation	2000	380
Complete meteor detection	2450	810

the hardware acceleration of the optical flow calculations improves the overall runtime by a factor of 3, the optical flow calculations alone are more than 5.2 times faster. This is a significant acceleration and allows the algorithm to process images in a reasonable time. The current processing rate of $\sim 0.8\,\mathrm{s}$ per image can be further improved: Currently, no other speed optimization is done, for example reducing the number of copy operations of image data. Furthermore, more functions could be accelerated by hardware, e. g. the blob detection. Thus, a processing rate of $\sim 0.5\,\mathrm{s}$ per imagee is realistic. Higher rates can be achieved with more powerful PLOC hardware.

5 SUMMARY and OUTLOOK

In this paper, we presented the concept of our novel meteor detection algorithm called *SpaceMEDAL*. It is based on optical flow calculations, in order to distinguish between meteor and background motion. This allows to solve the challenge of detecting meteors from a moving satellite.
The presented algorithm works well and is able to detect meteors of different properties, such as velocity and brightness, in front of different backgrounds. Applying *SpaceMEDAL* to our test set results in 94.68 % correct detections while 78.76 % of all meteors were detected. The high *F-measure* of 0.8599 indicates a very robust and reliable meteor detection algorithm.
The algorithm is a crucial part of any spaceborne meteor detection mission since the downlink capacity of satellites is limited. Thus testing and adapting the various parameters of the algorithm is important to ensure that meteors can be detected. Therefore, we developed the software *ArtMESS* to generate a systematic test set containing different meteors and different backgrounds. Furthermore, we developed and calibrated a Testbed, which displays those videos. Together with an automated imaging of the videos and automated evaluation of the algorithm results, this setup allows to test *SpaceMEDAL* fast, systematically, and in a realistic environment. The developed tools and procedures are a crucial part of the development effort. However, some work still needs to be done before *SpaceMEDAL* can be deployed. This includes work on the algorithm itself as well as on the implementation in the PLOC software. Regarding the algorithm, more tests are required using additional test sets in order to adapt the algorithm's parameters for different scenarios. The idea is to develop different parameter sets for different scenes. Therefore, test sets for different scenes should be generated and a wide variety of meteors implemented in those scenes. Depending on the scene (e. g. Ocean, clouds, or city lights) *SpaceMEDAL* should autonomously decide which parameter set should be used. This

could be done by evaluating the histogram of an image. Furthermore, the rotation of the satellite must be considered in more detail, thus test sets with different rotation rates should be developed and parameter sets derived.
Finally, the implementation in the PLOC software must be tested, to ensure the algorithm can be remotely commanded and works as intended. Once the implementation is finished, it is planned to publish *SpaceMEDAL* as well as the used tools to (e. g. *ArtMESS*) on the IRS git server (`https://egit.irs.uni-stuttgart.de/`).

REFERENCES

[1] Annika Stier et al. "Combination of Interdisciplinary Training in Space Technology with Project-Related Work through the CubeSat SOURCE". In: (2020) (cit. on p. 2).

[2] Peter S Gural. "Algorithms and software for meteor detection". In: *Advances in Meteoroid and Meteor Science*. Springer, 2007, pp. 269–275 (cit. on p. 3).

[3] Gunnar Farnebäck. "Two-Frame Motion Estimation Based on Polynomial Expansion". In: *Proceedings of the 13th Scandinavian Conference on Image Analysis*. LNCS 2749. Gothenburg, Sweden, June 2003, pp. 363–370 (cit. on p. 3).

[4] Marcel Liegibel et al. "Meteor observation with the SOURCE CubeSat - Developing a simulation to test on-board meteor detection algorithms". In: *4th Symposium on Space Educational Activities*. 2022 (cit. on pp. 5, 7).

[5] T Arai et al. "Meteor observation HDTV camera onboard the international space station". In: *Lunar and Planetary Science Conference*. 1777. 2014, p. 1610 (cit. on p. 5).

[6] Rafael Padilla et al. "A Survey on Performance Metrics for Object-Detection Algorithms". In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. 2020, pp. 237–242. DOI: 10.1109/IWSSIP48289.2020.9145130 (cit. on p. 8).

[7] Yutaka Sasaki. "The truth of the F-measure". In: *Teach Tutor Mater* (Jan. 2007) (cit. on p. 8).

[8] Julia Zink. "Test and optimization of an on-board meteor detection algorithm for the CubeSat SOURCE". English. Master Thesis. Stuttgart, 2021 (cit. on p. 8).

[9] John A. Nelder and Roger Mead. "A Simplex Method for Function Minimization". In: *Comput. J.* 7 (1965), pp. 308–313 (cit. on p. 8).

[10] SciPy documentation. *minimize(method='Nelder-Mead')*. `https://docs.scipy.org/doc/scipy/reference/optimize.minimize-neldermead.html\#optimize-minimize-neldermead`, last accessed on 27.10.2021 (cit. on p. 8).

[11] Jona Petri. "Satellite formation and instrument design for autonomous meteor detection". English. PhD Thesis. Stuttgart, 2022 (in review) (cit. on p. 9).

[12] Bastian Bätz. "Design and implementation of a framework for spacecraft flight software". Stuttgart, 2020. URL: `http://nbn-resolving.de/urn:nbn:de:bsz:93-opus-ds-112229` (cit. on pp. 9, 11).