Open-source meteor detection software for low-cost single-board computers

Denis Vida
The University of Western Ontario
55 PUBLICATIONS **95** CITATIONS
SEE PROFILE

Damir Segon
41 PUBLICATIONS **114** CITATIONS
SEE PROFILE

Peter Gural
Leidos, Inc.
71 PUBLICATIONS **677** CITATIONS
SEE PROFILE

Robert Cupec
University of Osijek
41 PUBLICATIONS **239** CITATIONS
SEE PROFILE

Open-source meteor detection software for low-cost single-board computers

Denis Vida[1], **Dario Zubović**[2], **Damir Šegon**[3], **Peter Gural**[4] **and Robert Cupec**[5]

[1] **Astronomical Society "Anonymus", B. Radić 34, 31550 Valpovo, Croatia**
Faculty of Electrical Engineering, University of Osijek, Kneza Trpimira 2B, 31000 Osijek, Croatia
denis.vida@gmail.com

[2] **Croatian Meteor Network**
dario@zubovic.email

[3] **Astronomical Society Istra Pula, Park Monte Zaro 2, 52100 Pula, Croatia**
damir.segon@pu.htnet.hr

[4] **Gural Software Development, 351 Samantha Drive, Sterling, Virginia, USA 20164**
peter.s.gural@leidos.com

[5] **Faculty of Electrical Engineering, University of Osijek, Kneza Trpimira 2B, 31000 Osijek, Croatia**
rcupec@etfos.hr

This work aims to overcome the current price threshold of meteor stations which can sometimes deter meteor enthusiasts from owning one. In recent years small card-sized computers became widely available and are used for numerous applications. To utilize such computers for meteor work, software which can run on them is needed. In this paper we present a detailed description of newly-developed open-source software for fireball and meteor detection optimized for running on low-cost single board computers. Furthermore, an update on the development of automated open-source software which will handle video capture, fireball and meteor detection, astrometry and photometry is given.

1 Introduction

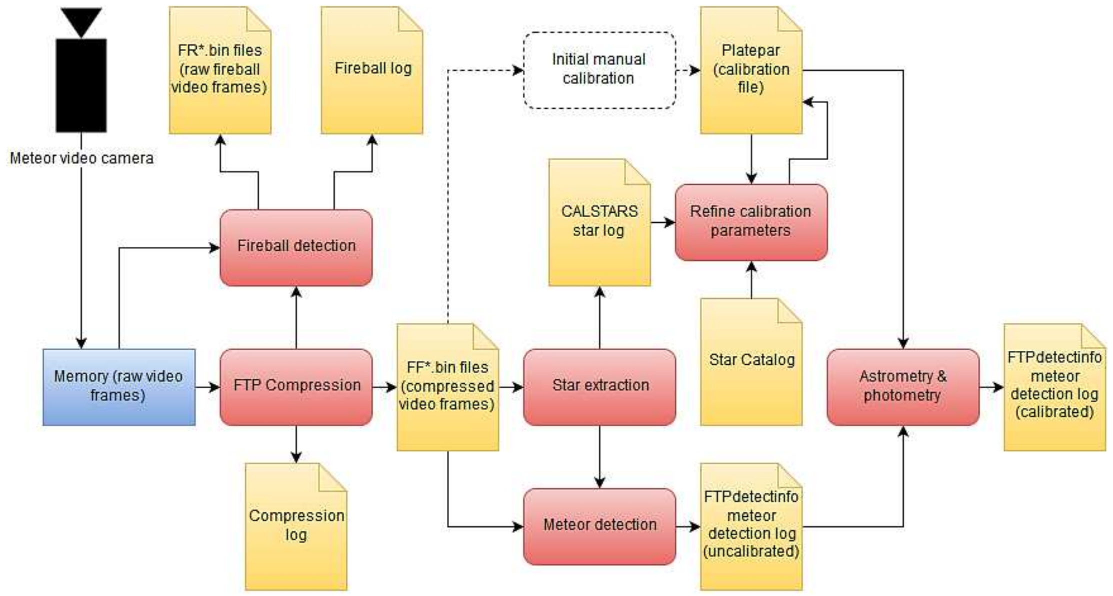
With the advent of low-cost sensitive video security cameras, amateur meteor enthusiasts quickly embraced this technology and noted its potential for meteor surveillance (Gural and Šegon, 2009). The technology has proven capable of delivering quality data, such that a meteorite recovery was possible based on an amateur meteor network (Borovička et al., 2015) using such cameras. As the price of these cameras has continued to decline, falling below 50 USD (Samuels et al., 2014), the main price component of a meteor station became the computer for recording and processing the data. As the computer's price is an order of magnitude larger than that of a single camera, the question of replacing it with a cheaper alternative naturally arises. The possible candidates were found in the form of low-cost single-board computers. A set of these computers were tested in (Zubović et al., 2015) and it was concluded that a viable alternative exists, namely the *Raspberry Pi 2* device which seemed affordable and powerful enough to serve the purpose. Furthermore, it was concluded that a low-cost meteor station using the *Raspberry Pi 2* computer could be built for about 150 USD, including all components. While the hardware is not the main topic of this paper, it is worth mentioning that a new generation of the *Raspberry Pi* has recently been released, the *Raspberry Pi 3* which offers up to 50% more computing power[1] than the previous generation. Although the hardware options are plentiful, there were no software solutions capable of running on such devices. Their ARM CPU architecture and the Unix-based operating system create a unique problem, where to successfully compile and use meteor processing pipeline software, one needs to obtain its source code and adapt it to run under such a system configuration. The work in (Zubović et al., 2015) presented software which works on *Raspberry Pi 2* devices, records video from the camera, compresses it into the CAMS FTP format (Gural, 2011) and performs a rudimentary real-time fireball detection.

In this paper new and improved algorithms are presented which include complete procedures for real-time fireball detection, meteor detection, star extraction and data calibration. All software is open-source and available on the project's GitHub page[2]. The authors believe that meteor surveillance is a matter of great importance, and as such it should be available to all. By making the software open-source, anyone can use the code, contribute to it and more experienced contributors can improve it significantly. The *Python* programming language was chosen as the main development language, while the computationally intensive parts of the code are written in *C++*. The choice of this combination of programming languages is common for astronomical purposes[3] in the recent years.

[1] https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/

[2] https://github.com/CroatianMeteorNetwork/RMS

[3] http://python-in-astronomy.github.io/



FR*.bin files (raw fireball video frames)
Fireball log
Initial manual calibration
Platepar (calibration file)
Meteor video camera
CALSTARS star log
Refine calibration parameters
Fireball detection
Memory (raw video frames)
FTP Compression
FF*.bin files (compressed video frames)
Star extraction
Star Catalog
Astrometry & photometry
FTPdetectinfo meteor detection log (calibrated)
Compression log
Meteor detection
FTPdetectinfo meteor detection log (uncalibrated)

Figure 1 – A block diagram of the developed software.

2 Software overview

An overview of the developed software is shown on *Figure 1*.

The raw frames from the grabber are compressed in the CAMS FTP compression format (Gural, 2011). The compression procedure compresses 256 raw video frames into 4 images: maximum pixel value image (maxpixel), average pixel value image excluding the maximum (avepixel), standard deviation pixel value image excluding the maximum (stdpixel) and an image which tracks the time of the occurrence of the maximum pixel value (maxframe). The time, i.e. the frame number of the maximum pixel value is encoded as a byte valued image level. If a certain pixel has several maximum values occurring at different frames, the frame number which is stored in the maxframe image is randomly chosen to distribute noise peaks uniformly in time.

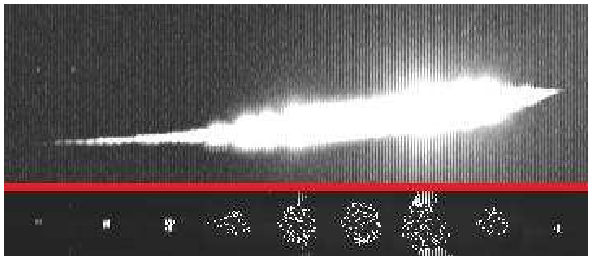


Figure 2 – Fireball maxpixel image (top) and several reconstructed frames showing compression artifacts (below).

While the aforementioned compression procedure works sufficiently enough to preserve meteors of moderate brightness, certain problems can occur with very bright fireballs. As the fireball saturates the CCD sensor and leaves a path of saturated values along its track, the compression algorithm only takes a single saturated value encoded in time. The effect of this is visible in *Figure 2*. The top of the figure shows the maxpixel image, while the bottom shows several reconstructed video frames. It can be seen that the reconstructed frames suffer from compression artifacts which can impair the precision of the centroid determination as information about the real position of the fireball is lost.

To counter this problem a real-time fireball detector was developed to enable saving the raw video frames while they are still in the temporary memory buffer. Fireballs have orders of magnitude higher signal-to-noise ratio in respect to the background than meteors of moderate brightness, and thus the authors believe that a dedicated fireball detector provides more consistent results than a combined approach. Furthermore, a fireball detection algorithm is simpler and faster, thus satisfying the real-time requirement. Fainter and moderate brightness meteors are detected on the compressed data with more elaborated methods offline, as they generally do not suffer from compression artifacts.

Besides fireball detection, to make a low-cost meteor station competitive with the existing solutions, an option to detect fainter meteors is highly desirable. As there are constraints on the computing power, the algorithm needs to be fast, reasonably robust and sensitive enough to detect meteors not detected by the fireball detector. To consider how fast the algorithm actually must be, Worst Case Execution Time and the maximum average time which can be spent per each image for meteor detection must be calculated.

To use the full power of the *RPi* computer, all 4 of its cores are employed. The capturing process has the highest priority and it needs to run in real-time, thus one core is completely dedicated to this task. Two cores are dedicated to video compression and real-time fireball detection. The compression and fireball detection process is serial, meaning that fireball detection is run after the compression. Thus each of these two cores run the same serial process, but the input data stream is alternated between the two, to double the available processing time per single serial procedure. The one remaining core runs star extraction and meteor detection. Once the capture process is finished, all 4 cores run star extraction and meteor detection procedures. Finally, after the detection procedures finishes, astrometry and photometry procedures are performed.

With the core utilization laid out, the total available processing time for star extraction and meteor detection can be calculated. The longest night of the year at latitude 50° north is about 16 hours, which translates into 5625 CAMS format FF files at 25FPS and 256 frames per file. With the proposed core utilization, during the longest night the algorithm will have 16 hours available during the night on one core, and the remaining 8 hours on 4 cores. Thus the total computation available is 48 hours (not taking into account system housekeeping and post-processing). This translates to the maximum average time which the algorithm can spend on one FF file of about 30 seconds. Furthermore, taking into account that the *Raspberry Pi 2* is at least an order of magnitude less powerful than contemporary normal-sized computers, the average maximum time the algorithm running on a normal-sized computer can spend on each image is around 3 second. As this time is quite short, the algorithm should make an effort to reduce the number of analyzed files early on.

3 Fireball detector

An initial version of the fireball detection algorithm was presented in (Zubović et al., 2015). In this paper an improved version of the algorithm is presented - the algorithm was re-implemented in *Cython*[4] for faster execution and input parameters were fine tuned. Furthermore, the algorithm is discussed in more detail below.

The input data to the detector are the FTP compressed file and raw video frames. To detect significant rises in image intensity over the background, image thresholding is performed with the following operation:

$$Threshold(max, avg, stddev) = \begin{cases} white, & (\max > avg + K_1 \cdot sttdev)\ \&\ (\max > 40) \\ black, & otherwise \end{cases}$$

The K_1 parameters determines how many standard deviations above the background level the maximum intensity must be to be considered to be part of the fireball, i.e. a white pixel. The chosen value was $K_1 = 4$, based on numerous experiments on fireball images. Furthermore, the minimum intensity level of the pixel must be 40. *Figure 3* shows the influence of the varying value of the K_1 parameter. As it can be seen on the given figure, values of K_1 below 4 produce too many white pixels. Values above 4 can produce black regions in the middle of the fireball – these are caused by the very high standard deviation values in the middle of the fireball. When these high values are multiplied by the K_1 parameter of a higher value, the $avg + K_1 \cdot sttdev$ expression can exceed the maximum digitization value of 255, thus making the maximum image value unable to pass this threshold. If one requires such a high threshold, it is possible to clip the calculated value to 254 and thus ensure saturated pixels get through. As this would introduce a slight overhead in computation time, this feature was not implemented.

Following this procedure, the image pixels are now either white (pixels of interest) or black (background). As the CAMS FTP format image contains information about the occurrence of every maximum pixel value, the thresholded image can be shown as a 3D point cloud, XY components representing the image axes, and Z component representing the time axis. The resulting 3D point cloud can be seen in inset *a) of Figure 4*. To further reduce the noise and reduce the total number of points of interest, a subsampling procedure is performed. The point cloud is sampled by 16 × 16 × 256 bins. A secondary thresholding is performed which is based on counting the number of points in each bin. If the number of points is less than 8, the bin is rejected. The result is shown in inset *b) of Figure 4*. As fireballs can often have bright flashes, an algorithm which removes the flashes (i.e. slices at the time axis which have considerably more points than average) is applied to the 3D point cloud. An example of such 3D point cloud is given on the inset *b) of Figure 4*, where a plane of points at frame 90 can be noticed. The flash filter looks for such planes in the data, i.e. the frames when the fireball 3D point cloud has at least 10x more points than the median number of points per frame. The frame is removed and the result is shown in inset *c) of Figure 4*.

After the described preprocessing, it is obvious from *Figure 4 inset c)* that the fireball is represented as a line segment in 3D space. Thus a new line segment detection algorithm has been independently developed and implemented in *Cython*. First, the points are sorted by their respective frames. Then the algorithm pairs each point with each other to hypothesize a line. Each hypothesized line is tested for the number of points in its neighborhood. The neighborhood is defined as cylinder around the hypothesized line (with a fixed radius). The algorithm evaluates each hypothesized line by the number of points and their distances from the line by producing a weighted score. The closer the point is to the line, the higher the score it has. Furthermore, if the algorithm determines that there is a discontinuity in the concentration of points, that particular hypothesized line is rejected as the goal is to find compact line segments. The line with the best cumulative score is chosen, its points are removed from the point cloud and the algorithm runs recursively until all acceptable lines are found. The pseudo code of the algorithm, as well as the description of the input parameters, are given in Code segment A. The line segment detection algorithm has a time complexity of O(N^3) in the worst case, and given

[4] http://cython.org/

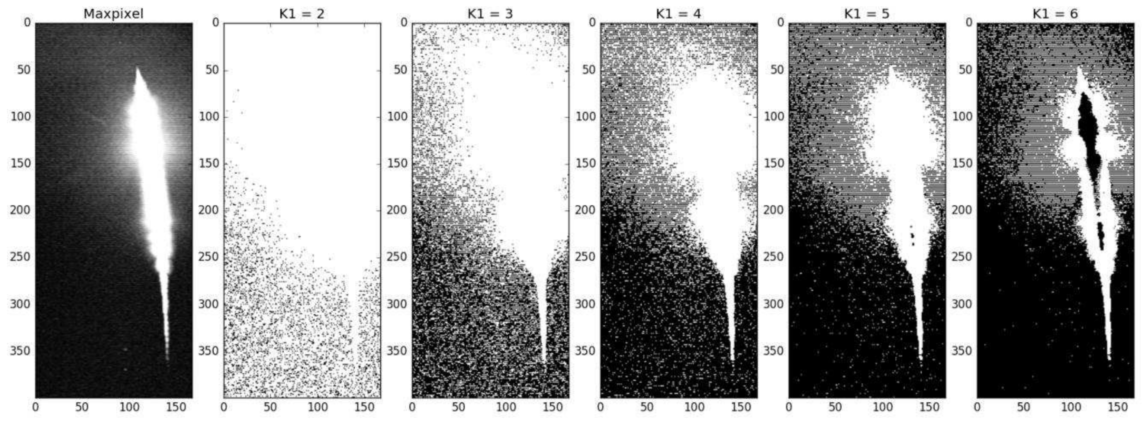


Figure 3 – Thresholding the fireball with various values of K_1.

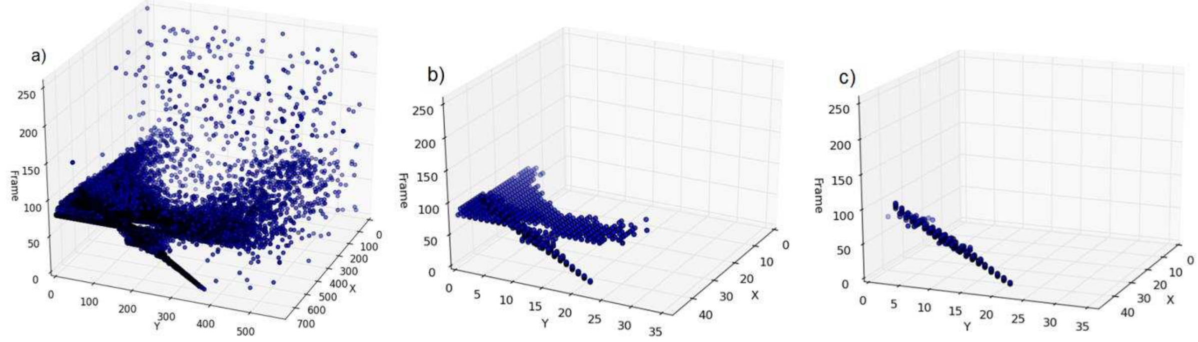


Figure 4 – Steps in the fireball detection thresholding procedure.

the restrictions on the computing time available, the total number of points which are fed into the algorithm was limited to 1000. If the point cloud contains more points, 1000 points are randomly chosen. In principle, the newly developed line segment detector is very similar to *RANSAC* (Fischler and Bolles, 1981), although it contains several key differences which enable it to search specifically for line segments, in contrast to unbounded lines. After the fireball is located, raw video frames containing the fireball are pulled from memory and are stored on disk for later use.

The performance of the fireball detector was evaluated on about a hundred examples of fireball images from the Croatian Meteor Network archives. The brightness of successfully detected events ranges from fireballs which saturated half the image, to 0^{th} magnitude meteors. After examining the results, the authors have concluded that the fireball detector is robust and suitable its purpose.

4 Meteor detector

To reduce the total processing time, the star extraction procedure (described in the next section) is run before meteor detection. If the number of detected stars is too low, meaning that the sky is not clear, the meteor detection algorithm will not be run at all on the given image. When the skies are clear, the processing flow will include the detection procedure.

As the CAMS compression format saves the maximum and the average value of each pixel during 256 frames, as well as its standard deviation, thresholding the image to find events brighter than the average is done by applying the following operation on the image:

$$Threshold(max, avg, stddev) = \begin{cases} white, & \max > avg + K_1 \cdot sttdev + J_1 \\ black, & otherwise \end{cases}$$

K_1 is a scaling factor which determines how many standard deviations above average the event should be, while J_1 is an absolute factor which adds to the total level threshold by adding a minimum background level. The combination of factors K_1 = 1.7, J_1 = 9 proved to be optimal for discriminating meteors from the background noise.

After the image is thresholded, the algorithm checks the ratio between the thresholded and the total area of the image. If the thresholded part covers more than 5% of the image, the image is rejected. The reason for this is that moonlit clouds can often cause many above average pixel exceedances, which in turn slows down the algorithm.

As one compressed image contains information of about 256 frames, this allows reconstructing the whole video from the compressed file. To reduce noise, the whole 256-frame block is not analyzed at once, but only a 64 frame "window" is reconstructed from the FF file. The

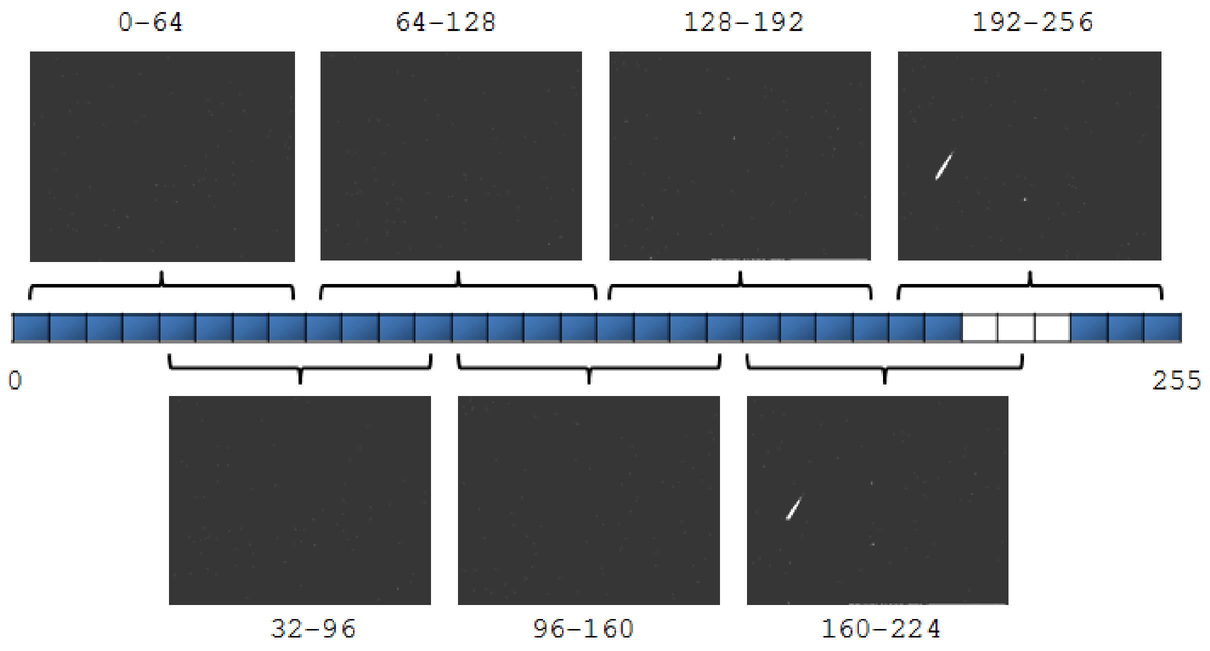


Figure 5 – Frame reconstruction and frame “windows”. The meteor appears on frames from 211 to 228.

starting frame of each reconstructed window is shifted by 32 frames, producing 7 such windows covering frame ranges of 1–64, 32–96, 64–128, 96–160, 128–192, 160–224, and 192–256, thus the windows are overlapped in time to avoid “leakage” of meteors spanning processing windows. The mentioned “window” is not a set of 64 actual frames, but the maxpixel of the short window block. *Figure 5* illustrates the described procedure and shows individual frame “windows”.

On each such window a set of image morphological operations (Gonzales and Woods, 2008) is performed. First, morphological cleaning is performed; a process which removes isolated pixels. This operation removes most of the noise on the image. *Figure 6* illustrates the described procedure.

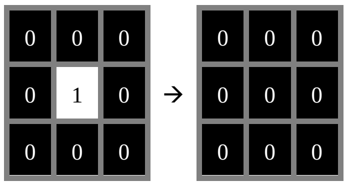


Figure 6 – Morphological cleaning.

Then a morphological bridging operation is performed which connects pixels which are on the opposite sides and all other pixels are 0. This operation helps to connect disconnected features on the image, such as broken lines. *Figure 7* illustrates the described procedure for 1 of 4 possible pixel orientations.

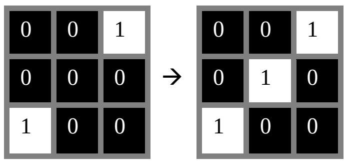


Figure 7 – Morphological bridging.

After that, a morphological closing is performed. Closing is a structured filling in of hollow image features which consists of two sub-operations: morphological dilation followed by erosion, using the same structuring element for both operations. This operation helps to fill in all the possible gaps in the thresholded meteor.

To prepare the image for line identification, all possible lines must be as thin as possible. Thus a *Zhan-Suen thinning algorithm* (Zhang and Suen, 1984) is applied to the image which skeletonizes the image i.e. makes all possible meteors on the image to appear as long thin lines.

Finally, a morphological cleaning is performed again to remove all noise on the image remaining after thinning. Now the image is ready to run the line detection algorithm. *Figure 8* shows an example of the maxpixel image (left), the image after thresholding (middle) and the image after the complete pre-processing procedure (right).

The image pre-processing procedure was implemented due to the peculiar operation of the chosen line finding algorithm. After a period of experimentation, it was decided to settle on the *Kernel-Based Hough Transform* (KHT) (Fernandes and Oliveira, 2008) due to its superior speed and performance, which was necessary due to the low computation power of single board computers. The authors of the *KHT* made it open-source which perfectly aligned with our needs and software development philosophy. The preprocessed images are fed into the algorithm and it returns all line candidates on the image.

After all lines have been retrieved on all 64-frame “windows”, similar lines are identified using the *Discrete*

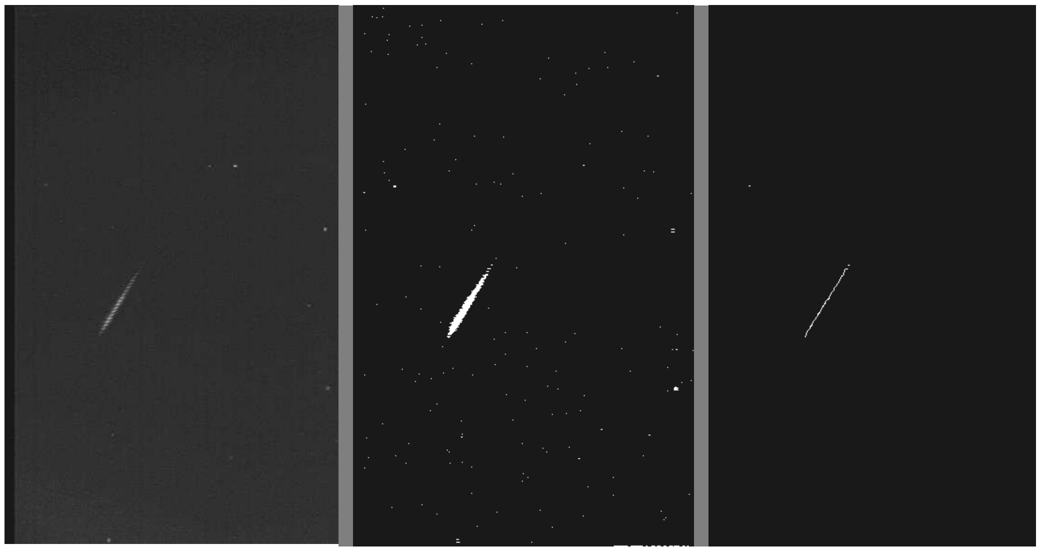


Figure 8 – Maxpixel image of a meteor (left), thresholded image (middle), image after preprocessing (right).

Fréchet distance (Eiter and Mannila, 1994) as the similarity measure and are averaged. During the line segment merging, the exact "windows" on which the line appears are tracked, thus the approximate time of the line appearance is known. In this point the algorithm has a list of candidate lines which need to be confirmed as meteors. If this list is empty, meaning no lines satisfying the given parameters were found, the procedure is aborted and the image is rejected and it is considered to not contain any meteors. On the other hand, if there are lines in the list, the algorithm proceeds to confirm that the found lines could be meteors.

The next phase of the algorithm determines if the candidate line contains a possible meteor by determining if the line propagates through time. First, as the approximate time of the line appearance is known as a range of frames between which the candidate line appeared, this fact is used to reconstruct the "window" image using the given frame range. Then a strip of about 50 pixels in width is extracted around the line. In CAMS FTP format, each pixel has an assigned time component of its maximum value during the 256 frame period, meaning that each pixel in the strip is given a time component. Thus a 3D point cloud is obtained - a line propagating through time should be a compact line in this point cloud, thus the same algorithm as the one in fireball detection is used, although with a different set of parameters to allow for smaller lines to be detected. The algorithm determines the exact starting and ending frames of the propagating line, as well as the true orientation of the line. Any event shorter than 4 frames (i.e. 0.16 seconds at 25 frames per second) is rejected due to a large number of such short events detected during cloudy weather, which can considerably slow down the algorithm. This also means that all meteors shorter than 4 frames are not detected. In the case of future improvements in available computational power, this restriction can be easily lifted.

After the algorithm determines the exact duration (i.e. the beginning and ending frames) of the event, centroiding is performed by reconstructing each frame of the event and again extracting a strip around the event. A center-of-mass calculation is performed, using pixel intensities as weights (Berry and Burnell, 2005). As the video camera employed produces an interlaced signal, a deinterlacing procedure is performed beforehand – centroiding is done separately on odd and even image rows, thus giving a half-frame time resolution. Finally, the obtained centroids are filtered by rejecting those which considerably deviate from the fitted trend line. *Figure 9* shows the marked centroids of the meteor shown on *Figure 8*. The results of the detection procedure are written out as a CAMS *FTPdetectinfo* file format, so that the results can be processed with the existing (although proprietary) CAMS procedures.

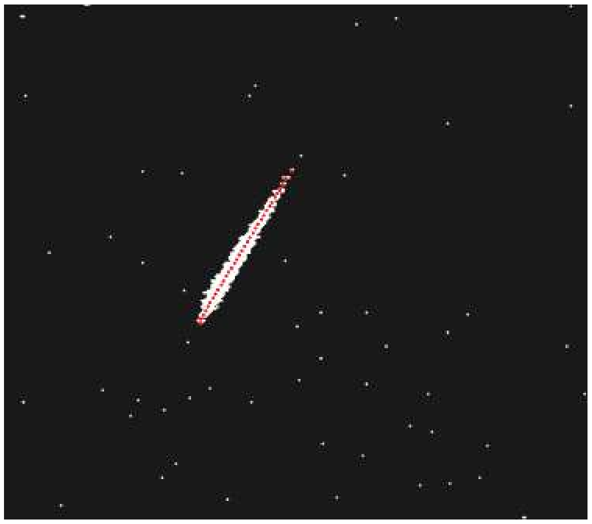


Figure 9 – A detected meteor with marked centroids.

The performance of the meteor detector was evaluated on about a hundred carefully chosen meteor images. The

Table 1 – Comparison of meteor detection performance between the new detector and the CAMS detector.

Type	Night ID	No. files	Total proc. time (sec)	Time per file (sec)	Meteors detected		False positives	
					New	CAMS	New	CAMS
1	VIB_20160419	3211	29390	9.15	14	11	53	18
2	OSE_20160417	3269	7770	2.37	0	2	9	61
3	VID_20160417	3263	37100	11.36	0	0	1679	304
4	OSO_20160419	3233	9430	2.91	2	4	36	2941
5	OSE_20160501	2950	2590	0.87	0	0	0	318
				TOTAL	16	17	1777	3642

goal was to sample a wide variety of meteors of varying brightness, duration and velocity to test the algorithm's detection performance. The algorithm's parameters were tuned until all chosen meteors were successfully detected.

Furthermore, the detector was tested on 5 full nights, each containing about 3000 individual image files. The night types were chosen to be representative of the conditions encountered during the year:

1. A clear and Moonless night with several meteors;
2. A cloudy night with the presence of the Moon with very few meteors;
3. A night with fast moving clouds with the presence of the Moon;
4. A rainy night in a light polluted environment, resulting in visible falling raindrops;
5. A cloudy and stormy night with the presence of lightning.

The goal of these data was to test the algorithms robustness and false positive rate. The results were compared to those obtained by the *MeteorScan detector* employed as a part of the CAMS processing pipeline (Jenniskens et al., 2011) in *FTP_CaptureAndDetect version 1.6* software. The results of comparison for full nights are given in *Table 1*.

Compared to the CAMS detector, the total number of false positives was considerably smaller. It was discovered that this was caused by the condition that the detection procedure is run only when a minimum number of stars is present on the image, thus eliminating most of the detections on clouds and during daytime. On the other hand, the new detector produced lots of false positives when part of the image contained fast moving moonlit clouds while the other part was clear. This behavior will be addressed in the future by introducing cloud mitigation techniques.

The total number of detected meteors was also smaller. After careful comparison, it was determined that the missing detections are those meteors shorter than 4 frames in duration (which are automatically rejected by the new detector) and meteors between the clouds. On the other hand, during clear nights the new detector performed similar to the CAMS detector. Real differences cannot be determined without a detailed comparison, but it is worth mentioning that in several cases the new detector detected more meteors than the CAMS detector (with the detection parameters used by the Croatian Meteor Network). Furthermore, on all tested data the algorithm never exceeded the maximum average processing time per image. The maximum average processing time per image for the tested nights on the *Raspberry Pi 2* device was about 11.5 seconds, including both the star extraction and meteor detection.

Although the detection rate was similar to the CAMS detection procedure, further analysis is needed. But based on these early results, it can be concluded that under the circumstances and the given computational power the newly developed detector is performing satisfactory for the needs of an amateur meteor enthusiast. Room for improvement still exists and it is hoped that a more successful algorithm will be implemented in the future, most probably the one given in (Gural, 2016). Furthermore, the algorithm should be tested on even more data to confirm its performance.

As the system is fully automated by design, a manual meteor confirmation procedure is not a part of the processing pipeline. False meteor detections will be rejected during orbit estimation as they will not form realizable orbit solutions. Nevertheless, as the results are CAMS compatible, it is possible to perform manual confirmation using the available software solutions, such as the *CMN_binViewer* (Vida et al., 2014).

5 Star extraction

To astrometrically calibrate the intrinsic (field distortion) and extrinsic (coordinate transformations) parameters of the camera, a set of stars from each recorded image is needed. Thus a robust algorithm for detecting stars on the recorded FF files was developed. The algorithm takes the "average pixel" image from the FF file and first calculates the mean intensity of the image. To quickly check if the algorithm should proceed at all, the mean image value is compared to a predefined threshold. If the image is too bright (e.g. an image recorded during the day), it is rejected. If the image passes this test, the inverse hyperbolic sine function is applied to all pixels on the image to adjust levels of the image so that the stars become more prominent. The maximum image filter (i.e. morphological erosion) is applied to the image; while on a copy of the original image a minimum image filter (i.e. morphological dilation) is applied. The difference of the 2 images thresholded by a fixed threshold value leaves only the areas of the image which are considerably brighter than their background. These peaks are detected and the center of mass is calculated for each on the original average pixel image, giving the approximate coordinates of the candidate stars.

To refine the results, and to better determine if the candidate is really a star or not, *point spread function (PSF) fitting* is performed by fitting a 2D Gaussian function to an area of 9×9 pixels centered around each candidate star using the least squares regression. The authors are aware that a 2D Gaussian does not perfectly represent the real *PSF* of the star, but for the purposes in video meteors which have a lower photometric resolution, a pure Gaussian *PSF* is assumed. Initial *PSF* parameters are approximated beforehand so that real stars converge quickly to a solution. Thus if the fitting does not converge in a limited number of iterations, the candidate is rejected. This procedure has proven to be a good discriminator between real stars and spurious detections. Furthermore, if the *PSF* fitting procedure is completed successfully, the covariance matrix of the *PSF* is evaluated. If the *PSF* is too narrow, the candidate star is rejected as a hot pixel (i.e. bright dot defect). Finally, as a consequence of fitting the *PSF*, the location of each star is known very accurately and its precision is on a subpixel level. The intensity of each star is calculated as a volume under the fitted *PSF*.

Finally, the stars found are written in the CAMS *CALSTARS* format, so that the calibration procedure can be done using CAMS-compatible procedures if needed.

The results of the new algorithm were compared to the results of the *CAMS FTP_CalStarExtractor* software. It was concluded that the newly developed algorithm yields very little false positives, only about 5%, while the *FTP_CalStarExtractor* often detects more false positives than real stars. Furthermore, the proposed algorithm yields virtually no detections during cloudy weather, thus its results can be used to determine weather conditions in the time of recording. When comparing the number of true positives between the two algorithms, the new algorithm detects about 90% of stars present in the CAMS data. The average number of detected stars per image during the periods of clear skies in the sample moderate field of view data was about 30. Combining frames from the same camera over the course of the night yields an average total number of detected stars in the tens of thousands.

6 Astrometry and photometry procedures

To transform the image coordinates of the meteor detection to celestial coordinates, an astrometric plate solution of the associated camera is needed. The initial plate constants (field center, scale, field distortion parameters) are first manually estimated by knowing the pointing direction of the camera and its optical properties. To further refine the plate constants, the detected stars need to be matched with stars from a star catalog. For this purpose, the Yale Bright Star Catalog[5] is used. To have a better quality of the solution and to cover a larger part of the focal plane, stars detected on images all throughout the night are used. As the total number of all stars in a single night can be in the tens of thousands, which can be hazardous for the computational time needed to calculate an astrometric solution, a random sample of images is taken where images with more stars have a greater probability of being chosen. At least 500 stars are needed to continue with the calibration, the number being chosen on the basis of findings in (Šegon, 2009).

The image coordinates of the chosen stars are transformed to celestial coordinates using the initial calibration parameters. The transformed coordinates are then matched to their nearest neighbors among the catalog values in celestial coordinates, but only if the coordinates are closer than a predefined angular distance threshold. The distance and the direction of the shift between each of the matched stars are recorded, the median values are calculated and the correction is applied to the plate constants. The procedure is repeated by reducing the angular distance threshold during each iteration, until the desired match is achieved. The matching metric is evaluated as a quotient of the standard deviation of the shift between the detected and catalog stars and the total number of matched stars. Thus a better solution is one that yields a smaller value.

Once the initial parameter refinement is complete, a more elaborate refining of the field center position is performed using the *Nelder-Mead method* (Nelder and Mead, 1965). Right ascension and declination of the center is adjusted until the algorithm converges to a stable solution – the same evaluation method is used as in the initial refinement procedure. Next, the distortion parameters are also refined using the same above-mentioned procedures. The image distortion is estimated by 3rd order polynomials with 2 extra “radial distortion” terms in both X and Y directions, albeit with different coefficient values:

$$f_x(x,y) = x + a_1 + a_2x + a_3y + a_4x^2 + a_5xy + a_6y^2 + a_7x^3 + a_8x^2y + a_9xy^2 + a_{10}y^3 + a_{11}x\sqrt{x^2+y^2} + a_{12}y\sqrt{x^2+y^2}$$

$$f_y(x,y) = y + b_1 + b_2x + b_3y + b_4x^2 + b_5xy + b_6y^2 + b_7x^3 + b_8x^2y + b_9xy^2 + b_{10}y^3 + b_{11}x\sqrt{x^2+y^2} + b_{12}y\sqrt{x^2+y^2}$$

The extra terms in the polynomials were first used as a part of the Croatian Meteor Network calibration procedures, but have been unpublished until now. During the initial development of the CMN procedures it was found that the modified polynomials produce smaller residuals compared to the ordinary 3rd order polynomials. This hypothesis has been tested again before the final implementation in this software and it was found that the

[5] http://tdc-www.harvard.edu/catalogs/bsc5.html

proposed equations produce significantly smaller standard deviations in the fitted star positions than the ordinary third order polynomials. The theoretical background behind the reasons of such behavior was not explored, that will be a topic of some future work.

After the astrometry parameter estimation is done and if the calibration was successful, a photometric calibration procedure is performed. Instrumental intensities of matched stars are compared to the apparent visual magnitude catalog values of said stars. A regression procedure is performed to fit the well-known intensity vs. magnitude function (Berry and Burnell, 2005):

$$m_1 = -2.5\log_{10} C_1 + 2.5\log_{10} C_2 + m_2$$

where m_1 is the calculated magnitude, C_1 is the input intensity, while C_2 and m_2 parameters are fitted from the abovementioned data. It is worth noting that the photometric procedures are very basic and with no regard to spectral sensitivity of the camera. Furthermore, no correction for saturated pixel values is performed. This part of the calibration procedure requires further work and improvement, which is hoped to be done in the future. Finally, meteor detections are converted from the image plane coordinates to celestial coordinates using the estimated plate constants and intensities are converted to apparent magnitudes.

The results of the proposed astrometric calibration procedure were compared to the results of the existing CMN calibration procedure. The new algorithm produced an order of magnitude better results, although results of the subsequent runs on the same dataset varied slightly because of the random sampling of the images from which the stars are used.

7 Discussion

The authors believe it is worth discussing the benefits that an automated low-cost meteor station could provide. Lowering the starting price of a meteor surveillance system would mean that existing networks could be easily expanded, as the human resources and a certain level of expertise exists among already organized groups. Furthermore, new networks could be easily formed with very little financial investment, meaning that meteor science would be available to a wider audience, especially in less than well-off nations. The total effect would be a considerable rise in the atmospheric collecting area and longitudinal coverage. An educational aspect should also be considered – students could be introduced to astronomy, computer and data science by installing such a system on their school and make them involved in every step of its operation. Moreover, scientists from other fields could recognize the practicality of a self-contained system with a video camera and repurpose it for their needs, such as bird watching or monitoring atmospheric phenomena.

If the project is favorably seen by a larger audience willing to set up a network of such systems, the authors believe that data produced by this hypothetical network using open-source software should be publically available. The usual arguments for keeping the data closed, such as the cost of the developed system, no longer justifies not publishing detailed data in this case, and no time is spent on manual processing as the system is fully automated. A similar open database exists in the form of the IMO Fireball Report (Hankey and Perlerin, 2014) and the authors hope that in the future more video meteor data will be open and the methods of its generation will become more transparent.

Finally, the benefits of a wide-spread meteor network to actual meteor science could be immense. Most meter-scale impactors are not observed optically, as the existing fireball and meteor networks cover only a fraction of the sky and most are only detected with non-optical methods which lack astrometric precision and show certain biases towards faster objects (Brown et al., 2016). In the recent years there have been reports of short meteor shower outbursts which were observed only by one or two meteor networks, namely the February Eta Draconds (Jenniskens and Gural, 2011) and April alpha Capricornids (SonotaCo et al., 2014). These occurrences lead to the question whether some meteor shower outbursts were not noticed due to overcast weather or the nonexistence of a meteor network beneath the skies where the outburst was visible.

8 Conclusion

A complete open-source software solution for video meteor capture and detection on the *RaspberryPi 2* single-board computer has been developed and described in detail. First, a set of requirements were set which such a station should meet. Next, real-time compression to CAMS FTP format and a fireball detection algorithm were described. Also, a newly-developed meteor detection algorithm was described and evaluated with the conclusion that it suits the needs of a low-cost meteor station. The pseudo code of a line segment detector in a 3D point cloud used by both fireball and meteor detectors was given. Furthermore, a star extraction algorithm which uses a Gaussian *PSF* fitting to stars was developed and tested with very positive results. Finally, the astrometry and photometry procedures were implemented and discussed.

While the individual segments of software described in this paper performed within the requirements on sample data, system tests during an actual night of meteor recording still need to be performed. Also, the software needs to be made more user-friendly and the documentation is to be expanded. It is the hope of the authors that the number of contributors to this project will rise in the future and that the developed system will find its place among meteor enthusiasts.

References

Berry R. and Burnell J. (2005). “The Handbook of Astronomical Image Processing”, Willmann-Bell, 2nd edition.

Borovička J., Spurný P., Šegon D., Andreić Ž., Kac J., Korlević K., Atanackov J., Kladnik G., Mucke H., Vida D. and Novoselnik F. (2015). “The instrumentally recorded fall of the Križevci meteorite, Croatia, February 4, 2011”. *Meteoritics and Planetary Science*, **50**, 1244–1259.

Brown P., Wiegert P., Clark D. and Tagliaferri E. (2016). “Orbital and physical characteristics of meter-scale impactors from airburst observations”. *Icarus*, **266**, 96–111.

Eiter T. and Mannila H. (1994). “Computing discrete Fréchet distance”. Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria.

Fernandes L. A. F. and Oliveira M. M. (2008). “Real-time line detection through an improved Hough transform voting scheme”. Pattern Recognition (PR), Elsevier, 41:1, 2008. Pages 299–314.

Fischler M. A. and Bolles R. C. (1981). “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. *Communications of the ACM;* **24,** 381–395.

Gonzalez R. and Woods R. (2008). “Digital Image Processing”. Pearson, Third Edition, pages 627–676.

Gural P. S. (2011). “The California All-sky Meteor Surveillance (CAMS) System”. In Asher D. J., Christou A. A., Atreya P. and Barentsen G., editors, *Proceedings of the International Meteor Conference*, Armagh, Northern Ireland, 16-19 September, 2010. IMO, pages 28–31.

Gural P. (2016). “A Fast Meteor Detection Algorithm”. In Roggemans A. and Roggemans P., editors, *Proceedings of the International Meteor Conference*, Egmond, the Netherlands, 2-5 June 2016. Pages 96–104.

Gural P. and Šegon D. (2009). “A new meteor detection processing approach for observations collected by the Croatian Meteor Network (CMN)”. *WGN, Journal of the IMO*, **37**, 28–32.

Hankey M. and Perlerin V. (2014). “IMO Fireball Reports”. In Rault J.-L. and Roggemans P., editors, *Proceedings of the International Meteor Conference*, Giron, France, 18-21 September 2014. IMO, pages 160–162.

Jenniskens P., Gural P. S., Dynneson L., Grigsby B. J., Newman K. E., Borden M., Koop M. and Holman D. (2011). “CAMS: Cameras for Allsky Meteor Surveillance to establish minor meteor showers”. *Icarus*, **216**, 40–61.

Jenniskens P. and Gural P. S. (2011). “Discovery of the February Eta Draconids (FED, IAU#427): the dust trail of a potentially hazardous long-period comet”. *WGN, Journal of the IMO*, **39**, 93–97.

Nelder J. A. and Mead R. (1965). “A simplex method for function minimization”. *Computer Journal*, **7**, 308–313.

Samuels D., Wray J., Gural P. S. and Jenniskens P. (2014). “Performance of new low-cost 1/3" security cameras for meteor surveillance”. In Rault J.-L. and Roggemans P., editors, *Proceedings of the International Meteor Conference*, Giron, France, 18-21 September 2014. IMO, pages 66–73.

Šegon D. (2009). “How many stars are needed for a good camera calibration?”. *WGN, The Journal of the IMO*, **37**, 80–83.

SonotaCo, Shimoda C., Inoue H., Masuzawa T. and Sato M. (2014). “Observation of April alpha Capricornids (IAU#752 AAC)”. *WGN, Journal of the IMO*, **42**, 222–226.

Vida D., Šegon D., Gural P. S., Martinović G. and Skokić I. (2014). “CMN_ADAPT and CMN_binViewer software”. In Rault J.-L. and Roggemans P., editors, *Proceedings of the International Meteor Conference*, Giron, France, 18-21 September 2014. IMO, pages 59–63.

Zhang T. Y. and Suen C. Y., (1984). “A Fast Parallel Algorithm for Thinning Digital Patterns”. *Communications of the ACM*, **27**, 236–239.

Zubović D., Vida D., Gural P. and Šegon D. (2015). “Advances in the development of a low-cost video meteor station”. In Rault J.-L. and Roggemans P., editors, *Proceedings of the International Meteor Conference*, Mistelbach, Austria, 27-30 August 2015. IMO, pages 94–97.

Code segment A. 3D line detector pseudo code

```
Function FindLines(Point_cloud, Lines_found){

  // Check if the previously found lines exceed the maximum number of lines to be found
  If (Length(Lines_found) >= Max_lines){
    Return Lines_found;
    }

  Results_list = [];

  For each point P1 in Point_cloud{
    For each point P2 in Point_cloud{
      Line = Line defined by P1 and P2;
      Distance_sum = 0;
      Point_counter = 0;
      Previous_P3 = P1;

      For each point P3 in Point_cloud{
        // Check if the point is close enough to the line
        If (Distance(Line, P3) < Distance_threshold){

          // Check if the point is too far away from the previous point
          If (Distance(Previous_P3, P3) > Gap_threshold){
            // Reject the hypothesized line if the previous point
            // was too far away from the second point that defines the line
            If (Distance(Previous_P3, P2) > Gap_threshold){
              Point_counter = 0;
              }
            Break loop;
          }
          Point_counter++;
          Distance_sum += Distance(Line, P3);
          Previous_P3 = P3;
        }
      }

      // Reject the hypothesized line if it envelops too few points
      If (Point_counter < Minimum_points)
        Continue loop;

      Average_distance = Distance_sum / Point_counter;
      Quality = Point_counter - Distance_weight * Average_distance;

      Add Line in Results_list;
    }
  }

  // Choose the best hypothesized line
  Best_line = Line with the largest Quality in Results_list;
  Point_ratio = (Number of points in Best_line) / (Number of points in Point_cloud);

  // Remove the points of the best line from the point cloud
  Point_cloud = Point_cloud \ Points(Best_line);

  // Add the best line to results only if it covers a minimum number of frames
  If (Frame_range(Best_line) >= Minimum_frame_range)
    Add Best_line in Lines_found;

  // Iteratively find lines on the point cloud until most of points
  // in the cloud have been covered, the remaining number of points is not too low,
  // and the flag for returning just one line was not set
  If ((Point_ratio < Ratio_threshold) & (Number of points in Point_cloud > 10)
       & NOT Return_one_line)
    FindLines(Point_cloud, Lines_found);

  Return Lines_found;
```

Table 2 – 3D line segment detector input parameters.

Name	Data type	Description
Point_cloud	list	a list of points in the point cloud, each point is defined by the (X, Y, Z) tuple
Max_lines	integer	the maximum number of lines which the algorithm should find
Distance_threshold	float	the radius of the cylinder around the hypothesized line
Gap_threshold	float	the maximum distance between subsequent points which make the line
Minimum_points	integer	the minimum number of points a line should have to be accepted
Distance_weight	float	the weight by which the point-line distance will be multiplied, a larger value of the parameter yields compact lines with a smaller amount of points, while a smaller value yields dispersed lines with more points
Minimum_frame_range	integer	minimum length of the Z axis component, i.e. the minimum number of frames the line segment covers
Ratio_threshold	float	minimum ratio between the found and the total points in the point cloud until the algorithm is stopped, e.g. if the ratio is 0.7, the algorithm will run until at least 70% of points are joined to a certain line, or no line satisfies the minimum requirements to be accepted
Return_one_line	boolean	if True, the algorithm will not do an iterative search, but return only one line

Table 3 – 3D line segment detector output description.

Name	Data type	Description
Lines_found	list	a list of lines found in the point cloud

Queuing to get some speaking time with Korado Korlević (left). Sirko Molau waiting for Denis Vida to finish…