ISON DATA ACQUISITION AND ANALYSIS SOFTWARE

Vladimir Kouprianov[(1)]

[(1)] *Central (Pulkovo) Observatory of Russian Academy of Sciences,*
196140, 65-1, Pulkovskoye ave., Saint Petersburg, Russia, Email: v.k@bk.ru

ABSTRACT

Since the first days of the ISON project, its success was strongly based on using advanced data analysis techniques and their implementation in software. Space debris studies and space surveillance in optical are very unique from the point of view of observation techniques and thus infer extremely specific requirements on sensor design and control and on initial data analysis, dictated mostly by fast apparent motion of space objects being studied. From the point of view of data acquisition and analysis software, this implies support for sophisticated scheduling, complex tracking, accurate timing, large fields of view, and undersampled CCD images with trailed sources. Here we present the historical outline, major goals and design concepts of the standard ISON data acquisition and analysis packages, and how they meet these requirements. Among these packages, the most important are: CHAOS telescope control system (TCS), its recent successor FORTE, and Apex II – a platform for astronomical image analysis with focus on high-precision astrometry and photometry of fast-moving objects and transient phenomena. Development of these packages is supported by ISON, and they are now responsible for most of the raw data produced by the network. They are installed on nearly all sensors and are available to all participants of the ISON collaboration.

1 INTRODUCTION

The International Scientific Optical Network (ISON) project [1] dates back to early 2000s when the first trial satellite observations where performed in several Russian astronomical observatories, independently of the former Soviet Union space surveillance system and using the existing telescopes, charge-coupled device (CCD) cameras, and custom-made software. At the end of 2004, Pulkovo Cooperation of Optical Observers was established as a mostly self-supporting volunteer project joining together professional and amateur astronomers and engineers, mainly from the post-Soviet space and those having long-standing contacts with Russian astronomers, in order to create an all-purpose independent coordinated worldwide network of optical facilities suitable for various tasks of observational astronomy that may benefit from using a large number of small-aperture but fast-response telescopes with large fields of view (FOV).

At that time, a software suite for observatory control and image acquisition has been developed already by the author at Pulkovo observatory and used for several years, mainly for near-Earth asteroid (NEA) follow-up [2]. This software appeared to be sufficiently versatile to suite the needs of the new project.

In 2004, the author also began creating a new general-purpose software platform for astronomical image analysis, Apex II [3,4]. The first application of the platform was in the field of NEA research as well. In 2005, a first version of the Apex-based package for satellite and space debris image analysis has been issued. Over the years, the package became the primary tool for initial analysis of different types of observations done by the ISON network and also acquired numerous features specifically targeted at space surveillance and tracking (SST).

Fig. 1 contains a very basic outline of the normal ISON data flow shown from the "low-level" point of view of the optical sensor. From this perspective, the data analysis center located at Keldysh Institute for Applied Mathematics (KIAM) in Moscow is a source of the latest orbital catalog and a schedule, for each sensor independently. The main and only output of the sensor is a set of tracklets of all orbital objects detected in the images acquired according to the schedule; these are sent back to the KIAM center which is responsible for their correlation, and which maintains the catalog and schedules observations for all sensors according to the latest data. Details of the latter process are completely out of the scope of the present report where we concentrate only on the flow of data within an individual sensor.

Sensor data flow starts from the telescope control system (TCS) which is a collection of software components that can be thought as a means to convert the schedule to a set of images – pixel arrays accompanied by metadata (information on the image environment necessary for data reduction and analysis). TCS operates the various observatory hardware and coordinates its individual components to work together for optimally solving the

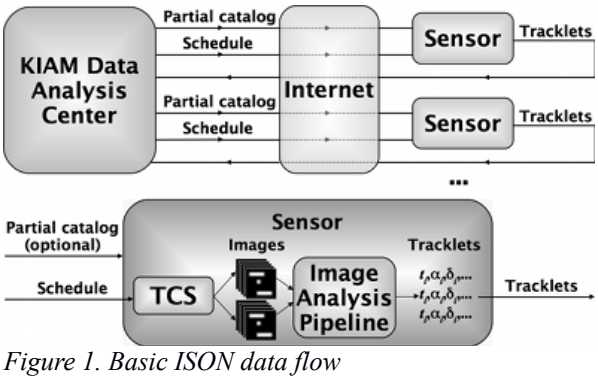

Figure 1. Basic ISON data flow

requested observation task. The second major sensor software component is an image analysis system that converts stacks of images into the ultimate sensor "product" – a set of individual detections' parameters assembled into tracklets. Here we focus on these two software components.

2 IMAGE ANALYSIS PIPELINE

Most of the ISON image analysis pipelines – and all pipelines targeted at SST – are built on top of the Apex II platform. The following describes the main design drivers and features of Apex II specific to space surveillance.

2.1 APEX II: Motivation and Design

Apex II image analysis system was initially an answer to the demand for a highly automated and accurate software for astrometric reduction of large amount of observations in the near-Earth asteroid (NEA) follow-up program conducted at Pulkovo [2]. However, the main design concept when starting development in 2004 was to create a flexible multi-purpose open-source software platform aimed at being suitable (at least, potentially) for any kind of data analysis in astronomy, with particular focus on optical and infrared imaging and on astrometric accuracy. The latter requirement was rarely met then in the widely used general-purpose astronomical image analysis software like IRAF, MIDAS, or IDL; the full range of algorithms for accurate differential astrometry, including precise measurement of centroid positions and means to correct instrumental and atmospheric distortions, was implemented only in the highly specialized software for several astrometric projects. The packages mentioned above, as well as several others, had also a long history of development and thus were based on the software design concepts that were often deprecated and inconvenient for modification and adaptation to non-standard purposes; some of them were commercial and closed-source, which limited their use.

Apex II pretended to fill this gap and still remain a modern highly versatile data analysis tool not limited to classical astrometry and NEA studies. To achieve this, we have chosen to create a package based on the open-source tools such as the Python programming language (www.python.org) and its extensive collection of packages for scientific computing that served as building blocks for a large library of astronomical image analysis algorithms and applications. Over the recent years, this platform acquired a constantly growing recognition throughout the scientific community. Fig. 2 illustrates how these ideas were implemented in Apex II; details on the particular software components are provided in [2,3].

Using Apex II in the ISON framework is a major challenge to its capabilities. A very special mode of observations dictated by the fast apparent motion of objects under study with respect to field stars infers very specific requirements on the image analysis software that should

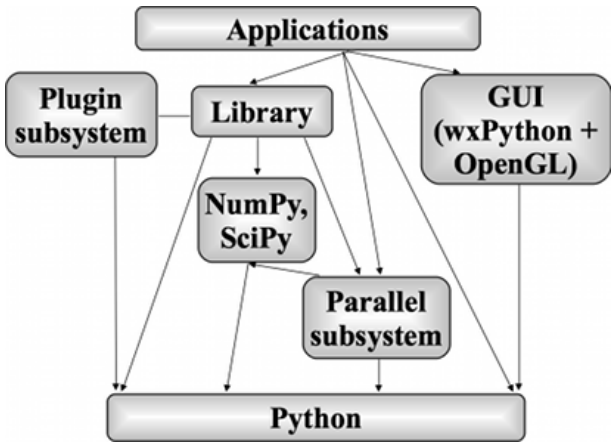

Figure 2. Structure of the Apex II image processing system

be able to deal with highly extended (trailed) images of field stars and/or Earth-orbiting objects. One more set of peculiarities comes from using mostly wide-FOV and hence extremely fast, down to $f/1$, optical systems in SST, which leads to the numerous negative effects of optical distortions and undersampling. Moreover, the highly heterogeneous nature of ISON that includes sensors of very different apertures and other optical characteristics and that are installed at places with very different sky conditions that are often far from perfect implies that the software should be extremely flexible and customizable and contain the appropriate algorithms for any possible case. Finally, the software should be able to process high amounts of data in short time and should ensure maximum reliability of space object detection with a minimum loss of sensitivity compared with the human eye.

To meet these requirements, Apex II, since the issuance of the first version of the satellite pipeline in 2005, acquired algorithms for robust non-stellar object detection based on mathematical morphology, substantially increased its performance by extensively using parallel computing, incorporated advanced initial orbit determination (IOD) and tracklet linking methods, and thus finally became a tool that produces most of the millions of ISON measurements of space object positions and fluxes per year. Its core and library are open-source and available from the author on a request; however, the SST package and all relevant algorithms are currently distributed only among the ISON members and cannot be disclosed elsewhere.

2.2 APEX II: Key Features for Space Surveillance

As it was noted above, using Apex II in SST seriously challenges its capabilities. First of all, a demand for real-time data analysis for any data rate leads to an absolute necessity of parallel computing. Fortunately, many image analysis algorithms are naturally parallelizable. Then, for all but the shortest exposure times, the shapes of images of Earth-orbiting objects and those of field stars are clearly distinct, which leads to a natural

application of mathematical morphology methods for object detection. The point-spread function (PSF) fitting technique inherent to the normal Apex II data flow using an efficient parallel implementation is a major source of the final accuracy of positional measurements. Finally, an effective *kd*-tree based algorithm for linking separate candidate detections into tracklets, similar to the one used by the PanSTARRS team [5], helps to quickly correlate multiple frames in a cadence and exclude false detections.

2.3 Parallel Computing

The first implementation of the Apex II parallel subsystem was based on the built-in Python multiprocessing capability which allowed to split calculations into multiple central processing unit (CPU) cores of the local workstation. This approach is simple and straightforward, and it allows also to run the pipeline on a supercomputer that has the capability to distribute local processes across multiple nodes for maximum scalability. The implementation is two-level: on the lower level, it splits the data flow for a single image into several subflows (e. g. parallel filtering of several parts of the pixel array or parallel PSF fitting for all detections); on the upper level, several images are processed in parallel. Since not all parts of the single-image pipeline are equally parallelizable, this combined approach allows for the maximum possible utilization of all available CPU cores when processing large datasets.

Apart from that, a newer implementation is currently under development in collaboration with the TFRM team at University of Barcelona. It is based on the OpenCL technology (www.khronos.org/opencl) which allows to run certain algorithms both on multiple CPU cores and on the general-purpose graphics processing unit (GPGPU) hardware [6]. Pixel processing is the most natural application for GPGPU programming; however, as it is noted in [6], a number of less-evident applications like e. g. optimization may also benefit from it. Combined with the older multiprocessing-based approach applied at the level of parallel reduction of multiple images, the OpenCL-based backend is expected to further increase the performance of Apex II SST pipelines and open the way for advanced uses like detection of faint space debris beyond the normal sensitivity limit of the sensor by means of shift-stacking series of images in all directions [7].

2.4 Mathematical Morphology

Traditional approaches to detection of fast-moving objects in stacks of CCD images usually rely on either taking difference images to eliminate everything that retains its position and brightness in adjacent images (i. e. field stars and other deep-sky objects) or comparing positions of all detections in the images to identify objects moving according to the linear pattern. However, using the first method in ground-based observations is strongly limited by atmospheric seeing that affects brightness and shape of stellar images and makes them look slightly different even in images taken close in time, thus leading to numerous artifacts left by subtraction. This method is also absolutely infeasible when dealing with images acquired in non-sidereal tracking mode (i. e. for almost any practical type of space debris observations), as trailed stellar images do not preserve their pixel positions and therefore are not eliminated by subtraction.

The second group of methods relies on analyzing measured pixel or sky positions of all detections in multiple images and is hence less sensitive to atmospheric noise, since a cutoff may be inferred on the jitter of coordinates across images to avoid false movers. The drawback of this approach, however, is the need to first accurately measure positions of all detections in all images, down to the sensitivity limit, which may become very time-consuming.

Our approach that is implemented in the ISON image analysis pipeline, first described in [3], overcomes this difficulty by using binary morphological filtering to eliminate star streaks before detection and thus considerably reducing the amount of work to measure and analyze positions. The basic idea behind this method is that the shapes of star streaks are fully defined by pixel scale, exposure duration, seeing, and tracking rate, and are thus quite accurately known beforehand and are clearly distinct from the shapes of space debris images. Then, using methods of mathematical morphology and a properly constructed filter kernel, one can detect image features corresponding to star streaks and finally eliminate them.

A somewhat similar approach is presented in [8]. Unlike our technique that is based on binary morphology, the iterative matched filter described there does effectively the same but acting on grayscale images to gradually wipe out star streaks. This should have an advantage of potentially higher sensitivity and the ability to detect space debris overlapping with star streaks, at the expense of computation time and a certain loss of reliability due to amplification of point-like artifacts like cosmic ray hits, hot pixels, and noise peaks. However, we have not done a direct comparison of both methods yet and cannot definitely conclude on their relative performance.

Candidate space debris positions that are generated by the Apex II morphology-based moving object detector are then passed to the *kd*-tree based tracklet linking algorithm mentioned in Section 2.2 that also identifies and removes the remaining spurious detections. Final tracklets are validated against IOD and sent back to the KIAM data analysis center, as shown in Fig. 1.

3 TELESCOPE CONTROL SYSTEM

Here we describe the design ad implementation of the second principal ISON sensor software component that is responsible for acquiring the imaging data.

3.1 TCS Software Requirements

Each of the thousands of small and large robotic telescopes in the world is controlled by its TCS software, the only ultimate goal of which being to obtain scientific data in whatever form it is possible for the given instrument. Even if we limit ourselves to optical imaging, there are numerous programs, open-source, proprietary, and custom-made (the latter, most often, on large telescopes), that are responsible for pointing and tracking, image acquisition, as well as for planning observations and controlling the various auxiliary observatory equipment. Among them, at least two – RTS2 (www.rts2.org) and INDI (indilib.org) – are open-source, support a wide range of the commonly-used astronomical instrumentation, and are adopted by many teams for "classical" astronomical applications involving deep-sky and slow-moving Solar system objects.

However, SST poses a number of important requirements that are rarely or never needed in other types of observations. Most of them are dictated by the relative proximity of objects to observer and thus by their fast apparent motion. The final accuracy of positional measurements is affected by timing which, depending on the orbit, should be perfect to several tens of milliseconds (high orbits) or to fractions of a millisecond (low orbits). To follow an object, the system should support fast and variable-rate tracking, either according to the ephemeris or with the feedback from real-time positional measurements. As it was noted in Section 2.1, SST requires wide-FOV optical systems, and it is often more cost-effective and flexible to use multiple parallel optical sensors rather than a single larger-FOV one. Also, to provide immediate follow-up of new discoveries for better IOD, many SST systems (including many ISON sites) are equipped with several optical sensors (e. g. survey, follow-up, and characterization subsystems) that work together and should be coordinated; a possible data flow for this case is given in Fig. 3. This leads to the need to simultaneously control complex setups with multiple sensors and to get feedback from the image analysis pipeline in real time. Finally, although not strictly required, a capability of dynamic rescheduling of observations that also takes into account sky conditions (incl. cloud coverage) may increase the overall performance, especially for sensors located at lower-quality astronomical sites.

Apart from the above conditions that are specific to SST, a good astronomical data acquisition software is expected to comply to a number of more generic requirements. First, a potentially large number of hardware components in the observatory that a single control computer cannot always accommodate demands for some type of distributed software architecture that would allow to easily reconfigure the observatory setup to split it across multiple control and data processing workstations. Then, a web-based user interface (UI) is required to easily access the TCS, monitor its state and progress of observations, and allow for manual intervention when necessary in a uniform manner, both locally

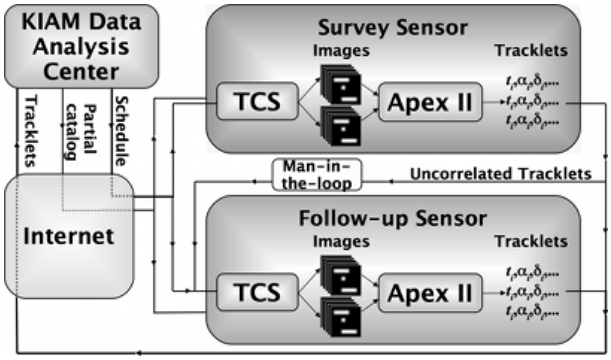

Figure 3. On-site follow-up

and remotely. A cleverly designed modular hardware support architecture and the corresponding application programming interface (API) are required for easily porting the TCS to any new hardware when upgrading the sensor, while an extensive datalogging capability not only helps to track operation errors but is useful for scheduling maintenance and replacement of certain hardware parts.

3.2 First Generation of the ISON Data Acquisition Software

As it was mentioned in Section 1, by the time of establishment of the ISON collaboration in mid-2000's, there existed a set of software applications and components for general-purpose telescope control, image acquisition, and other tasks supporting robotic observations. They were developed at Pulkovo Observatory by the author since the year 2000. Their design was driven mainly by the need in modular approach and flexibility, accurate hardware-assisted timing, and focus on the user interface that should be convenient, look familiar to professional astronomers, and provide easy access to most of the common operations. The software was oriented mainly towards the "supervised automated" mode of observations when the normal operation flow runs according to the schedule, while the detailed hardware state and progress of operation are easily available to the observer who has also the capability to manually override the automatic operation at any time.

The software [2] consists of the following large components. *CHAOS* package is an integrated TCS application that is responsible for basic scheduling, ephemeris support, and controlling the mount and other auxiliary equipment like dome, focusers, limit switches, sensors, etc. The list of supported mount controllers include Meade LX-200 compatibles, SynScan handpad, EQMOD-compatible controllers, Sidereal Technologies servo controllers, ASCOM-compliant devices, and several other less widely-used robotic mounts. *CameraControl* is an integrated application for CCD camera and filter wheel control and for image examination and storage. Most popular cameras and filter wheels, including FLI, SBIG, Apogee, and others, are supported. The *Datalogger* application provides system-wide logging of the TCS operation. And, finally, the hardware-disciplined *timing subsystem* provides a com-

mon API to non-realtime software events and to hardware triggering capabilities. The whole package is quite diverse regarding the programming languages and frameworks used (Delphi, C, Ada'95, Fortran, Python, etc.); inter-component data exchange is mostly based on dynamic linking using a set of binary APIs and on custom networking protocols.

This set of software components has been adapted to some most important ISON network needs and currently drives the vast majority of ISON sensors. Its advantage is its relatively long history (almost twice as long as the ISON's) and hence stability; it is also flexible enough to suite at least the most basic of the SST requirements listed above. However, all these packages are based on a deprecated software model, which strongly limits its extensibility potential, and it still lacks enough flexibility for some of the most advanced recent ISON observations strategies. The major problem is a weak integration with the image analysis facility (Apex II), which severely complicates the implementation of such modes as on-site follow-up (Fig. 3) and limits several other things like the ephemeris engine capabilities and maintaining accurate alignment and focus.

3.3 FORTE – a New TCS Software Package

To overcome the limitations described in the previous section, we have chosen to write a new integrated TCS and data acquisition software from scratch. The new package, called *FORTE* (Facility for Operating Robotic Telescope Equipment), is based on Python and thus shares the common platform with the image analysis infrastructure, which makes integration with Apex II very straightforward and natural. Using such a high-level and notoriously slow language as Python for controlling hardware – a task that inevitably contains time-critical code – is of course an arguable decision. However, as it is the case for Apex II, the most time-critical FORTE code that directly interfaces the hardware can be implemented in C and easily linked to Python level; Python itself, in turn, is used for scripting and controlling lower-level blocks, which is a classical way of using scripting languages. Writing most of the code in a higher level not only saves coding time; this also allows one to easily implement some very sophisticated algorithms like automatic alignment and automatic capturing of twilight flats, as well as to achieve a very high degree of configurability. Although any decent TCS software has a scripting capability of a certain kind, the latter is usually added to the existing low-level hardware control infrastructure, which gives one the level of control far inferior to that achievable by FORTE, where the very core of the TCS is implemented in the same language and at the same level of generality.

FORTE distributed core is based on a unique remote procedure call (RPC) mechanism that goes beyond such well-known Python RPC implementations as Pyro (pypi.python.org/pypi/Pyro4) and features transparent remote access even to such objects as error stack traces

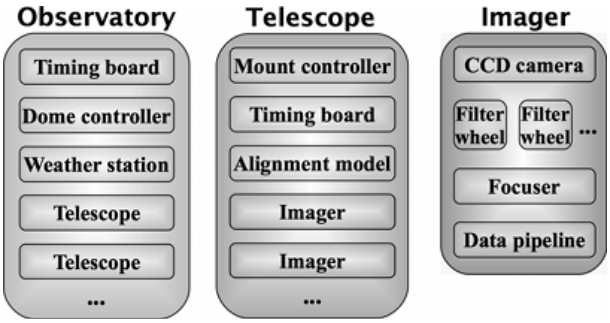

Figure 4. Observatory setup and its components

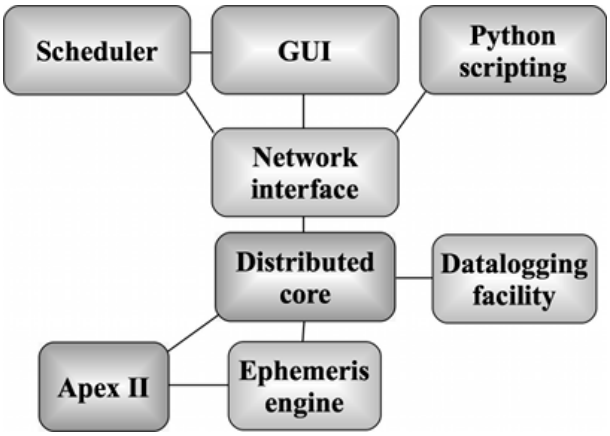

Figure 5. FORTE software components

and inter-process synchronization primitives. This allows one to spread hardware components across the network in a fully arbitrary and configurable manner. FORTE RPC uses two kinds of network transport for serialization: the internal binary protocol for maximum efficiency and the human-readable extensible markup language (XML) based protocol for maximum portability, e. g. implementing device driver modules and client applications in languages other than Python.

FORTE is designed for maximum flexibility and scalability. Fig. 4 illustrates the most generic observatory setup and its logical components. One can see that FORTE is able to control as simple setup as an amateur-grade goto mount plus CCD camera, as well as the recently constructed series of multi-dome and multi-telescope ISON sites sponsored by the ROSCOSMOS grant, by means of just changing the text-based configuration file. New hardware support modules are easily added according to simple APIs. Other software components of FORTE are shown in Fig. 5. Here we briefly describe them and highlight some specific FORTE features.

Forte *datalogger* is based on the built-in Python logging facility; thus it also automatically handles log messages from all external modules that use the same facility. Various backends are supported, including disk files with optional automatic rotation, Unix syslog daemon, Windows event log, and sockets. The actual logging configuration, including specifying destinations for different types of events and message formats, is fully defined by

the user. The same facility is used to collect the various hardware usage statistics, including motor revolutions, shutter cycles, voltages, and so forth.

Ephemeris engine provides the instantaneous object positions and velocities for pointing and tracking. Apart from its own ephemeris data sources, the engine has access to those supported by Apex II, including stellar catalogs, Solar system ephemerides, and databases of orbits of artificial Earth satellites and space debris. *Network interface* is the common gateway for externally controlling and monitoring FORTE operation via the XML-based RPC protocol. This is facilitated by several client applications, including web-based graphical UI (GUI) and console-mode clients, and by Python scripts for automation of certain routine observation tasks.

All hardware devices, as well as the top-level Observatory device (see Fig. 4), are always in one of the predefined *hardware states*. The *offline* state assumes that all devices are in safe state (e. g. telescope parked, dome closed, CCD thermoelectric cooler disabled, etc.) and ready for power-off; while in offline, the link between FORTE and hardware is not established. Devices are in the *suspend* state during long delays in the normal operation (e. g. due to unfavorable weather conditions or daytime); hardware link is established, and FORTE monitors the various hardware characteristics but does not take any active control actions. In the *standby* state, FORTE makes sure that devices are ready for immediately starting the normal operation but prevents any actual movement or other similar activities. Finally, the *online* state is the only one for normal operation, when FORTE accepts requests for doing observations.

An important FORTE feature is the *image pipeline* that is essentially a user-defined set of operations on the image data and metadata. Pipelines consist of elementary operations like image calibration, display, or storage, run sequentially, in parallel, or in any combinations. They are initiated asynchronously immediately after the image readout; metadata hold a set of TCS state parameters before, during, and after integration, as well as some accompanying information like weather conditions. A certain default pipeline is associated with each optical channel of the observatory, but it can be also overridden by client applications individually for each exposure. The most basic image pipeline consists of just storing the image on disk as a flexible image transport system (FITS) file; this is what most of the simple TCS packages do. A more complex example may involve on-the-fly image analysis of a set of images to detect tracklets and initiate follow-up observations on another sensor in case of uncorrelated detection.

Another noteworthy FORTE feature is its *event* system. Various events are generated by TCS components on certain state changes. The user can define actions for some relevant events by means of Python scripting. Some possible examples of "event – action" pairs are: the overall cloud coverage is above 90% → switch to standby; relative humidity is above 95% → switch to suspend; ambient temperature has changed by 10º → perform auto-focusing. FORTE event system makes a large contribution to its overall flexibility and, in the right hands, may become a powerful tool for building a very intelligent observatory.

For a long time, ISON used (and is still using) the old Russian space surveillance system format for raw measurement interchange. This format is very restrictive and is capable of storing only a very limited set of parameters, and with limited numerical precision. To avoid data loss during the interchange and to handle a more extensive set of parameters, including those related to object characterization, ISON is currently moving to a new XML-based format. Apart from being able to store data with precision appropriate to the actual accuracy of measurements, this format may contain extended parameters related to color photometry, shape of the objects' images, and the various primary accuracy estimates. This may also help to evaluate the reliability and quality of individual measurements. The new format that is fully supported by FORTE is also fully extendable. Below is an example of a measurement represented in this format:

```
<meas>
  <sensor>12345</sensor> <id>12001002</id>
  <filename>/.../25.20120101T001122345.fit</filename>
  <utc>2012-01-01T00:11:22.345678</utc>
  <ra_j2000>1.2345678</ra_j2000>
  <dec_j2000>-2.345678</dec_j2000>
  <ra_j2000_error>0.123</ra_j2000_error>
  <dec_j2000_error>0.234</dec_j2000_error>
  <mag>15.678</mag> <mag_error>0.05</mag_error>
  <snr>5.678</snr> <x>123.456</x> <y>789.012</y>
  <x_error>0.0234</x_error> <y_error>0.0345</y_error>
  <vel_ha>-0.123</vel_ha> <vel_dec>1.234</vel_dec>
  <length>39.7</length> <width>2.5</width>
      <rot>43</rot>
      ...
</meas>
<meas>
  ...
</meas>
```

4 CONCLUSIONS

We have described here the basic structure, design principles, and implementation details of the standard telescope control, data acquisition, and image analysis software currently driving almost all ISON sites. Among other factors, the resulting performance of ISON sensors was for a long time limited by non-realtime initial data processing and by weak integration of the image analysis pipeline with TCS. During the years 2010–2013, the Apex II image analysis platform acquired a parallel subsystem. Along with the use of mathematical morphology methods for fast-moving object detection, as well as the

kd-tree based tracklet linking algorithm, this resulted in much higher computational performance of initial data reduction. A new observatory control system, FORTE, is tightly integrated with the data reduction pipeline, which strongly enhances its capabilities and results in a significant improvement of space debris discovery rate and of the overall ISON performance in general.

5 REFERENCES

1. Molotov, I.; Agapov, V.; Titenko, V.; Khutorovsky, Z.; Burtsev, Yu.; Guseva, I.; Rumyantsev, V.; Ibrahimov, M.; Kornienko, G.; Erofeeva, A.; Biryukov, V.; Vlasjuk, V.; Kiladze, R.; Zalles, R.; Sukhov, P.; Inasaridze, R.; Abdullaeva, G.; Rychalsky, V.; Kouprianov, V.; Rusakov, O.; Litvinenko, E.; Filippov, E. (2008). International scientific optical network for space debris research. *Adv. Space Res*. **41**(7), 1022–1028.
2. Devyatkin, A. V.; Gorshanov, D. L.; Kouprianov, V. V.; Vereshchagina, I. A.; Bekhteva, A. S.; Ibragimov, F. M. (2009). Astrometric and photometric observations of solar system bodies with Pulkovo Observatory's automatic mirror astrograph ZA–320M. *Sol. Sys. Res.* **43**(3), 229–239.
3. Kouprianov, V. (2008). Distinguishing features of CCD astrometry of faint GEO objects. *Adv. Space Res.* **41**(7), 1029–1038.
4. Devyatkin, A. V.; Gorshanov, D. L.; Kouprianov, V. V.; Verestchagina, I. A. (2010). Apex I and Apex II software packages for the reduction of astronomical CCD observations. *Sol. Sys. Res.* **44**(1), 68–80.
5. Kubica, J.; Denneau, L.; Grav, T.; Heasley, J.; Jedicke, R.; Masiero, J.; Milani, A.; Moore, A.; Tholen, D.; Wainscoat, R. J. (2007). Efficient intra- and inter-night linking of asteroid detections using kd-trees. *Icarus* **189**(1), 151–168.
6. Fluke, C. J.; Barnes, D. J.; Barsdell, B. R.; Hassan, A. H. (2011). Astrophysical supercomputing with GPUs: critical decisions for early adopters. *Publ. Astron. Soc. Australia* **28**(1), 15–27.
7. Yanagisawa, T.; Nakajiama, A.; Kurosaki, H. (2005). Detection of small GEO debris using automatic detection algorithm. In *Proc. 4th European Conference on Space Debris, Darmstadt, Germany, 18–20 April 2005*.
8. Lévesque, M. P. (2011) Detection of artificial satellites in images acquired in track rate mode. In *Proc. AMOS-Tech. Conf., Wailea, Maui, Hawaii, 13–16 September 2011*, E66.