RAA 2018 Vol. **18** No. **5**, 54(14pp) doi: 10.1088/1674–4527/18/5/54
© 2018 National Astronomical Observatories, CAS and IOP Publishing Ltd.
http://www.raa-journal.org *http://iopscience.iop.org/raa*

Research in Astronomy and Astrophysics

LAMOST CCD camera-control system based on RTS2

Yuan Tian[1,3], Zheng Wang[1,3], Jian Li[1], Zi-Huang Cao[1,3], Wei Dai[2], Shou-Lin Wei[2] and Yong-Heng Zhao[1,3]

[1] Key Laboratory of Optical Astronomy, National Astronomical Observatories, Chinese Academy of Sciences, Beijing 100101, China; *tianyuan@bao.ac.cn*
[2] Computer Technology Application Key Lab of Yunnan Province, Kunming University of Science and Technology, Kunming 650500, China
[3] University of Chinese Academy of Sciences, Beijing 100049, China

Received 2017 November 24; accepted 2018 February 8

Abstract The Large Sky Area Multi-Object Fiber Spectroscopic Telescope (LAMOST) is the largest existing spectroscopic survey telescope, having 32 scientific charge-coupled-device (CCD) cameras for acquiring spectra. Stability and automation of the camera-control software are essential, but cannot be provided by the existing system. The Remote Telescope System 2nd Version (RTS2) is an open-source and automatic observatory-control system. However, all previous RTS2 applications were developed for small telescopes. This paper focuses on implementation of an RTS2-based camera-control system for the 32 CCDs of LAMOST. A virtual camera module inherited from the RTS2 camera module is built as a device component working on the RTS2 framework. To improve the controllability and robustness, a virtualized layer is designed using the master-slave software paradigm, and the virtual camera module is mapped to the 32 real cameras of LAMOST. The new system is deployed in the actual environment and experimentally tested. Finally, multiple observations are conducted using this new RTS2-framework-based control system. The new camera-control system is found to satisfy the requirements for automatic camera control in LAMOST. This is the first time that RTS2 has been applied to a large telescope, and provides a referential solution for full RTS2 introduction to the LAMOST observatory control system.

Key words: telescopes — techniques: imaging spectroscopy — methods: observational — instrumentation: detectors

1 INTRODUCTION

The Large Sky Area Multi-Object Fiber Spectroscopic Telescope (LAMOST, also named the Guo Shou Jing Telescope) is a meridian reflecting Schmidt (Wang-Su-type, Cui et al. 2012) telescope. LAMOST, as a Chinese national scientific research facility, is operated by the National Astronomical Observatories, Chinese Academy of Sciences (NAOC), and is located at Xinglong Station, which is a national facility open to the astronomical community. As the telescope having the highest spectral acquisition rate in the world, LAMOST has already successfully observed and processed more than 7.68 million spectra[1], and is a powerful tool for wide-field and large-sample astronomy (Zhao 2015).

The main spectroscopic survey instruments of LAMOST are 16 low-resolution spectrographs (Hou et al. 2010), each of which contains two scientific charge-coupled-device (CCD) cameras (English Electric Valve (EEV) 203–82, 4k×4k pixels); thus, 32 CCDs are used for observations (Zou & Wang 2006). In 2009, the present authors developed CCD control software to coordinate procedures and collect status information for the 32 CCDs (Deng et al. 2010; Deng et al. 2011). A schematic diagram of the spectrographs and CCDs is shown in Figure 1.

[1] *http://dr4.lamost.org*

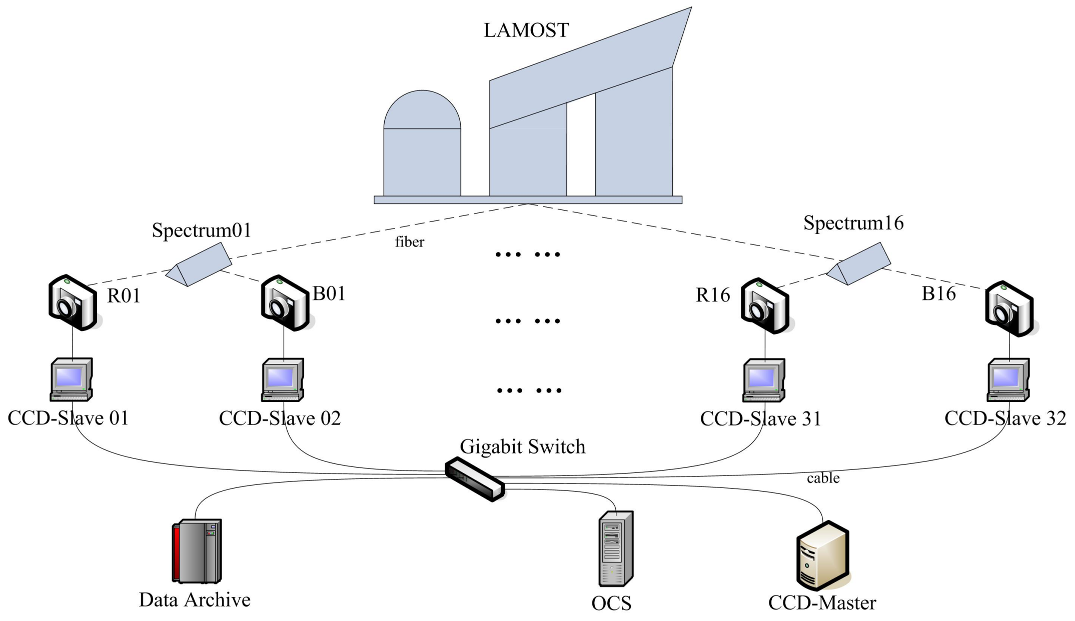


Fig. 1 Spectrographs and CCDs in LAMOST.

The developed software has worked for 8 years, and satisfied its basic design goal. However, this software is excessively complex, maintenance is tedious and full integration into the LAMOST observatory control system (OCS) is difficult. With the development of the LAMOST spectral survey, in order to improve software stability and observation efficiency, automation and robustness have become new requirements. It is apparent that the existing CCD control software cannot meet these challenges; thus, a software upgrade is required.

Computer-controlled telescope technology began to appear in the 1980s, following the introduction of personal computers (PCs). The considerable advancements in modern information technology since the beginning of the 21st century have contributed to its rapid development, and the goals of automation and robustness have now been achieved for a large number of telescopes. The development history of computer-controlled telescope technology can be divided into four stages: Automated Scheduled Telescopes, Remotely Operated Telescopes, Robotic Autonomous Observatories and Future Robotic Intelligent Observatories (Castro-Tirado 2010). Thus, the present study targets development related to the third stage, Robotic Autonomous Observatories.

Following an in-depth study of telescope automation control technology and software (Cui et al. 2013), we chose Remote Telescope System 2nd Version (RTS2), a representative third-generation control technology, for introduction to the LAMOST OCS. The first stage of this upgrade project involves implementation of RTS2 to drive the 32 CCDs, as one of the core OCS components. RTS2 is an open-source package for remote telescope control under the Linux operating system and is written in C++. It is designed to run in fully autonomous mode, which can be used to coordinate all autonomous scheduling, telescope pointing, data acquisition, instrument hardware monitoring, etc. (Kubánek et al. 2006).

The autonomous capabilities and robustness of RTS2 benefit from its object-oriented design and reasonable class hierarchy. With regard to communication, the code is divided into three levels: the communication code, device-type specific library and native drivers. The communication code is based on transmission control protocol/internet protocol (TCP/IP), using custom protocol and heartbeat technology, and can handle various network transport transactions and error recovery. The device-type library, which summarizes the common steps of the general telescope equipment, handles device commands (initiation of mount movement, exposure, etc.). The native drivers can drive various device types (CCDs, mounts, sensors, etc.). By inheriting the base device-type class and overriding parts of the virtual function interfaces, a software re-developer can quickly create a new dedicated device driver, which can then be customized for application to various tasks.

As a result of continuous improvement, RTS2 has evolved into a modular package supporting a variety of hardware devices and providing reliable general-purpose observatory control software. Thus, RTS2 is currently running on various small telescope setups and laboratory equipment control systems worldwide[2] (Kubánek et al. 2012b; Zhang et al. 2016). However, because of its complexity, RTS2 has not been successfully applied to large-aperture telescopes to date.

The aim of this study is to upgrade the existing LAMOST camera-control software to adapt to the RTS2 framework, as the first stage of the ultimate project aim of introducing RTS2 to the LAMOST OCS. Here, the developed RTS2-based camera-control system is described in detail and implemented in the actual environment, with multiple test observations being conducted.

2 SOFTWARE DESIGN ARCHITECTURE AND IMPLEMENTATION

As the 32 CCDs are the most important acquisition components of the LAMOST spectral equipment, their control system must be stable, automated and easy to maintain. To satisfy these requirements, we have introduced an RTS2-based software framework to the LAMOST CCD control system to upgrade the existing software. The CCD control system is shown in the context of the overall LAMOST software framework in Figure 2.

An object-oriented design method is employed for RTS2, which can support many kinds of cameras. A method for application of RTS2 to customized devices also exists[3], and this technology is very simple and convenient to use. In contrast, the existing LAMOST CCD control application is more complex. Note that the type of CCD camera employed by LAMOST is not supported by RTS2 directly. Further, no use case involving the control and coordination of such a large number of cameras by RTS2 has been reported to date. Therefore, it was necessary to design the system meticulously and to perform careful testing.

2.1 Design Architecture

In order to bridge RTS2 with the 32 EEV CCDs (real cameras) of LAMOST, the designed software is divided into three layers: the center controller layer (OCS, integrated with RTS2), virtual device layer (named "CCD-Master") and real control layer ("CCD-Slave" and Universal Camera, also named UCAM). UCAM is stand-alone driver software for scientific CCD cameras developed by Lick Observatory. Because of its good versatility and stability, LAMOST used it as the driver for EEV CCDs (Zou & Wang 2006; Jia et al. 2010). The software architecture is depicted in Figure 3.

In the center control layer, a customized RTS2 camera component (named "LAMOST-CCD") is created, which processes related RTS2 operations. The master-slave paradigm is used in the virtual device layer and real control layer. In the virtual device layer, a master, i.e., CCD-Master, is designed, which distributes commands to and collects status information from the slaves. It also maintains a connection with LAMOST-CCD, to receive command-and-response general status information. In the real control layer, CCD-Slave is implemented as a slave, which is a bridge connecting CCD-Master and the EEV CCD stand-alone driver software, i.e., UCAM.

This design scheme preserves the advantages of RTS2, e.g., target scheduling, control automation and fault tolerance. We also attempted to retain the existing software interface. Separation of the command/status stream was used to achieve a simple design, accelerating the code development progress.

2.2 Implementation of Customized Camera Class in RTS2 (LAMOST-CCD)

A new RTS2 camera class was created to control the LAMOST CCDs, named "LAMOST-CCD." This is an RTS2 camera module and inherits from rts2camd::Camera. The inheritance diagram of our camera class in the RTS2 framework is shown in Figure 4.

The rts2camd::Camera class is a generic camera module class provided by RTS2, being an abstract class for all kinds of cameras. It simply summarizes the general control steps and statuses of astronomy cameras and reserves a large number of (virtual functions) interfaces (Wei et al. 2014). In this project, LAMOST-CCD overrides a small number of virtual functions. These virtual functions can transmit special commands and statuses defined by the authors (for details on these custom commands, see Sect. 3).

The RTS2 framework implements connections between its modules using TCP sockets, and uses its own communication protocol. The core communication class is rts2core::Connection (Kubánek et al. 2008). To satisfy the communication requirement of RTS2, we created a connection class named "DevConnectionLAMOSTCCD," which inherits from rts2core::Connection. This class is used to manage the

[2] *http://rts2.org/wiki/obs:start*

[3] *http://rts2.org/wiki/doku.php?id=code:camera_driver*

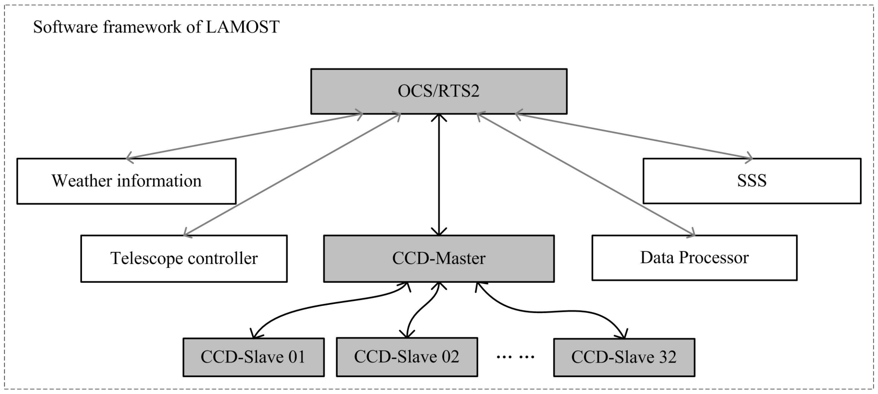


Fig. 2 Spectrographs and CCDs integrated into LAMOST (SSS: Survey Strategy System, which is responsible for generating observation plans and arranging observation sequences).

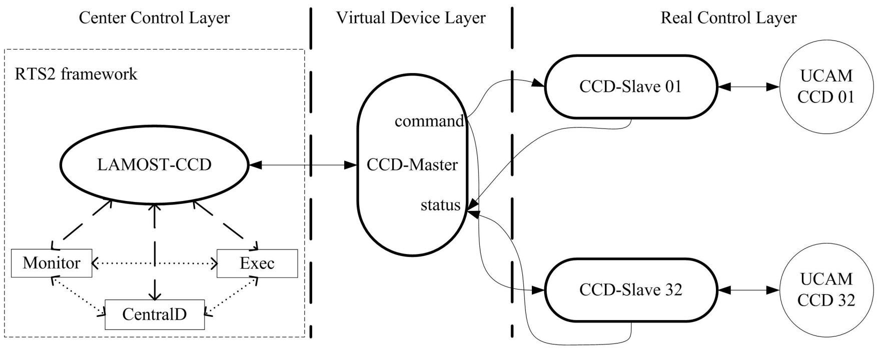


Fig. 3 Overall software architecture (CentralD: Name resolver and observatory housekeeper; EXEC: "Executor" device scheduler).

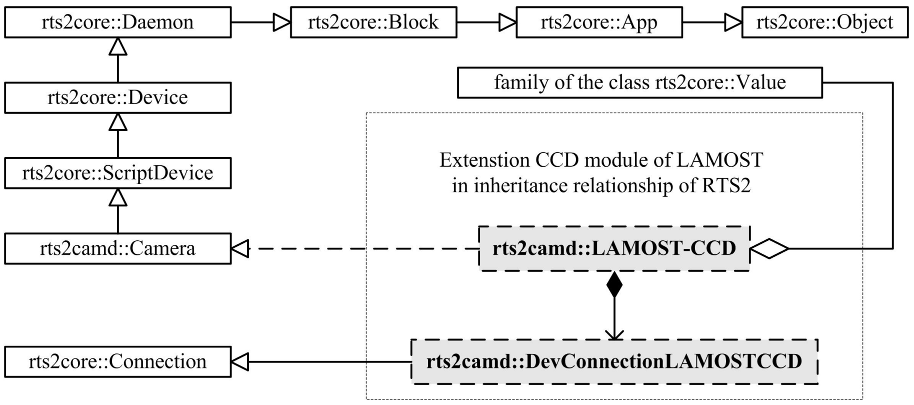


Fig. 4 Inheritance diagram of customized camera module class in the RTS2 framework.

connection between LAMOST-CCD and CCD-Master. We override its interfaces using functions such as processLine() and setState(). The detailed unified modeling language (UML) diagram is shown in Figure 5.

Following the RTS2 rules, when LAMOST-CCD runs on a separate process, it attempts to register itself into the RTS2 CentralD (the name resolver and observatory housekeeper) on rts2core::Connection. Meanwhile, it creates a DevConnectionLAMOSTCCD object to connect to CCD-Master. When this connection has been established, LAMOST-CCD adds this DevConnectionLAMOSTCCD object to its connections list. Subsequently, LAMOST-CCD functions in the same manner as any other RTS2 device module.

When the RTS2 executor or clients send a command to LAMOST-CCD, the latter transmits a command to CCD-Master through the TCP connection managed by DevConnectionLAMOSTCCD.

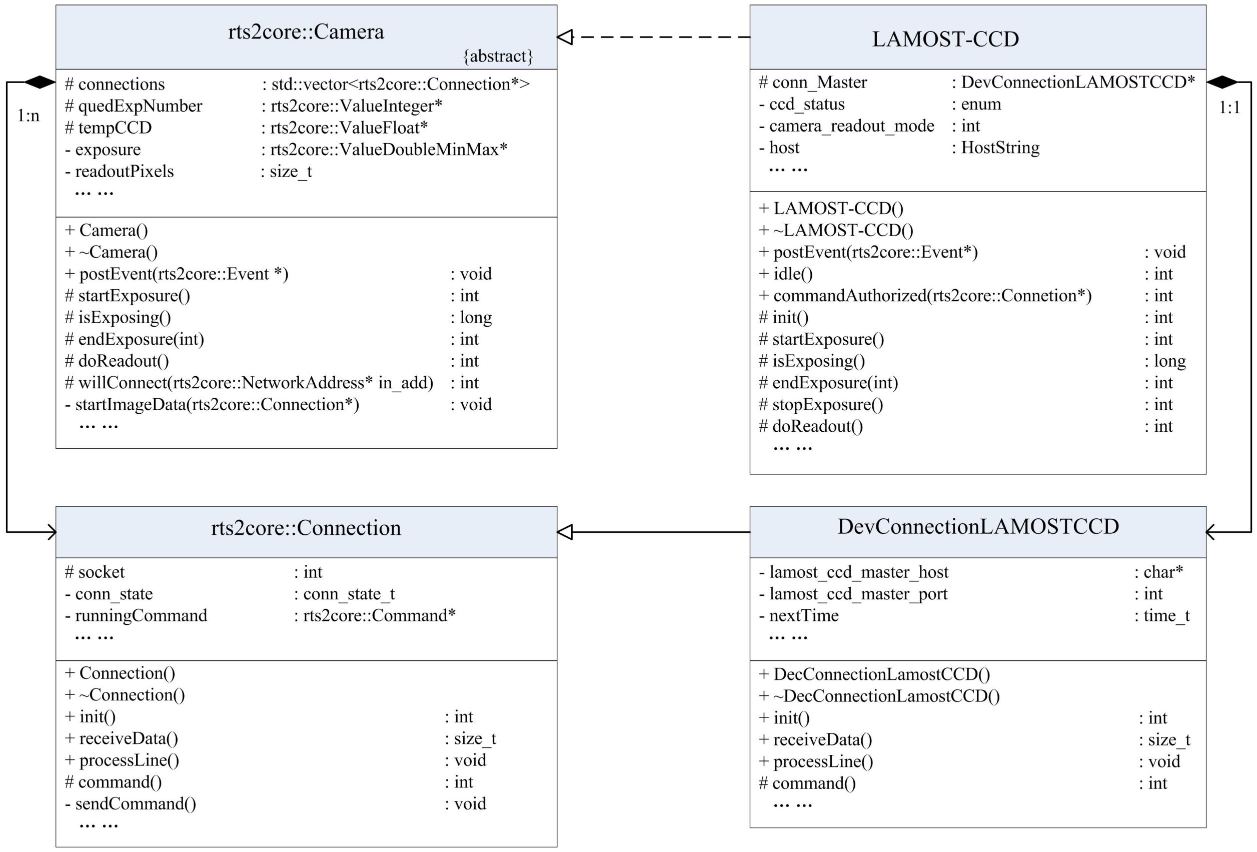


Fig. 5 LAMOST-CCD class UML diagram.

When CCD-Master responds with aggregate statuses, DevConnectionLAMOSTCCD translates these statuses into custom events (EVENT_LAMOST_EXPOSURE_START, EVENT_LAMOST_READOUT_END, etc.), and delivers these events to LAMOST-CCD. Finally, LAMOST-CCD, as an RTS2 device module, changes its own state and performs the appropriate post-processing.

As RTS2 provides a daemon-running framework, connection management framework and event handling mechanism, we can implement the LAMOST-CCD class very simply and rapidly.

2.3 CCD-Master

In order to map the 32 LAMOST CCDs to the RTS2 camera module, we added a virtual layer between them and implemented a master-slave program paradigm.

CCD-Master has two tasks. As the first task, CCD-Master (as a socket server) accepts connections and receives commands from LAMOST-CCD. Then, it processes these commands (translates, sets special parameters, etc.). Finally, it connects to the 32 CCD-Slaves (as a socket client) and distributes these commands. As the second task, CCD-Master (as a socket server) accepts connections and receives status messages from the CCD-Slaves. Then, it judges whether all cameras are working well. Finally, it provides a total status report to LAMOST-CCD as one virtual camera. The internal structure of CCD-Master is shown in Figure 6. As a simple synchronous mode through which socket communication is used to ensure that command and status transmissions are not blocked by each other, CCD-Master implements two processes separately, one for command and the other for status.

In an ideal scenario, CCD-Master should accurately distribute commands to the 32 cameras simultaneously, especially when the commands are "start/stop exposure." However, because of principles implemented for the program operation, this cannot be achieved very precisely. Fortunately, the nocturnal observations conducted at LAMOST typically involve long-term exposures (600 s − 1800 s), and the UCAM time sensitivity is 0.01 s. Therefore, a TCP-based multi-thread design plan for command distribution was adopted. This plan is sta-

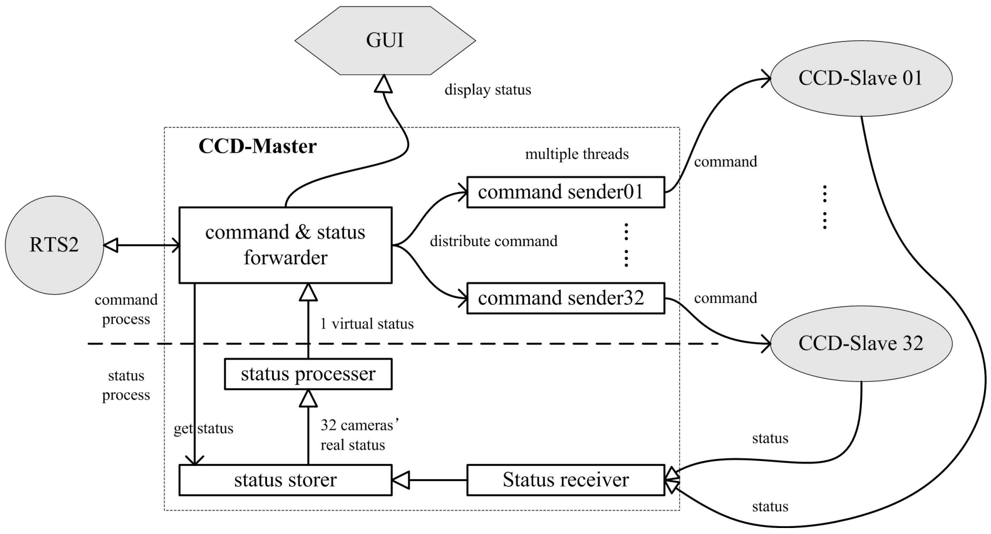


Fig. 6 CCD-Master internal structure diagram (GUI: Graphical user interface).

ble and low-cost, and satisfies the engineering requirements well. The time difference between multi-threads can be negligible in this case. Note that this approach was examined in the experiment in this study, and detailed results are presented in Section 4.1.

In the status process, one thread is used, because the status-collection time accuracy is not critical. We added a storer to cache the status results for the 32 cameras. When RTS2 sends command "Get Status," the CCD-Master's command and status forwarder (in the command process) transfer this command to the storer. Then the storer sends the status results for the 32 cameras back to the status processer, and the status processer generates a virtual status. Finally, the status sender obtains this virtual status and returns it to LAMOST-CCD (RTS2). The status reception performance of CCD-Master is reported in Section 4.2.

A graphical user interface (GUI) was implemented for CCD-Master, which provides observers with the option to obtain more detailed information on the real cameras. The GUI is shown in Section 4.3.

2.4 CCD-Slave

The internal structure of CCD-Slave is shown in Figure 7. This design can retain the original interfaces of the existing software. This concept is very important when upgrading the existing software of a large telescope that has been running stably for a long time. CCD-Slave is a bridge between CCD-Master and UCAM, which is located in the PC of each CCD controller.

Like CCD-Master, CCD-Slave also has two separate processes, one of which manages commands, while the other handles status messages. The command listener creates a socket and receives commands from CCD-Master, before translating the commands to a special string (or binary codes) that can be understood by UCAM. When CCD-Master sends a "Get Status" command, the command-listener status flag changes to TRUE. Then, the command listener connects to the CCD-Master status endpoint and sends the status information, which is cached in "status buffer."

The status collector process registers itself into the UCAM information channel and subscribes to certain status topics; then, it receives messages continuously. When a new status message is received, the status collector pushes it into the status buffer through the pipe, which is a bridge that connects the two processes. The pipe is created when CCD-Slave is initiated.

3 INTERNAL COMMUNICATION PROTOCOL

RTS2 employs its own communication protocol (Kubánek et al. 2008), which is based on sending American Standard Code for Information Interchange (ASCII) strings over TCP/IP sockets. It is fast, simple and robust. As a device module, LAMOST-CCD uses the existing RTS2 protocol to communicate with other RTS2 modules. To communicate with CCD-Master, LAMOST-CCD uses customized commands and statuses based on the RTS2 protocol. Table 1 lists various custom-defined commands and status messages.

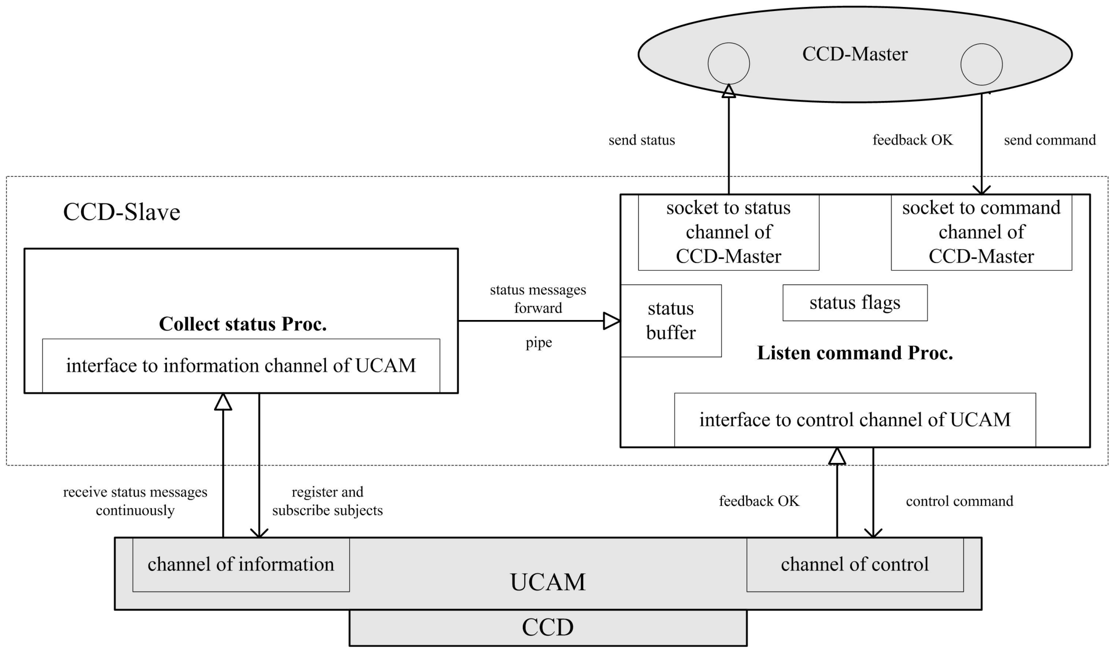

CCD-Master
send status
feedback OK
send command
CCD-Slave
socket to status channel of CCD-Master
socket to command channel of CCD-Master
status messages forward
pipe
status buffer
status flags
Collect status Proc.
interface to information channel of UCAM
Listen command Proc.
interface to control channel of UCAM
receive status messages continuously
register and subscribe subjects
feedback OK
control command
channel of information
channel of control
UCAM
CCD

Fig. 7 CCD-Slave internal structure diagram.

For convenience, we used a key-value pattern to transport and store various parameters. Implementation of this approach is very simple and this method is also very flexible. Using this technique, we could easily add and modify parameters throughout the entire code development process. For example, when LAMOST-CCD receives a "set parameters" command from the RTS2 internal module, the former then translates the command according to Table 1 and adds a "key-value" pair (e.g., <selected=all>). Then, LAMOST-CCD sends the translated command to CCD-Master. CCD-Master receives that command, parses it, obtains the value of "selected" and then forwards the command to all CCD-Slaves. Each CCD-Slave receives and parses the command and then drives its CCD camera.

4 SOFTWARE DEPLOYMENT AND TESTING

After software development was completed, the software was deployed in the real camera control environment of LAMOST (shown in Fig. 1). The 32 LAMOST cameras (EEV CCDs) obtain light signals from 4000 optical fibers. Each camera has a controller PC, which has an Intel(R) Core(TM) i3-2100 3.10-GHz central processing unit (CPU) and 4 GB of memory. The CCD-Master server has a 2-way Intel Xeon 2.53-GHz processor (containing 16 logical CPU cores in total) and a 12 GB memory. The hardware configuration of an RTS2 LAMOST-CCD PC is identical to that of the camera-control PC. All computers are connected to a Gigabit Switch and a CentOS 6.8 operating system is installed on each computer. We designed a series of experiments to test the performance of the new control software, to verify whether this software can satisfy the actual observation requirements for LAMOST. The results are reported in the following subsections.

4.1 Command Distribution Performance

We designed the first experiment to test the performance of the TCP-based command distribution, and to determine whether the new system satisfies the engineering requirements for controlling 32 real CCD cameras. Note that the performance of the existing User Datagram Protocol (UDP)-based system is also discussed in this subsection, for comparison with the upgraded system. We produced alphabetic character command data (8 B, 12 B, ..., 1 KB), and distributed them via both the existing software and the new system. To guarantee experimental accuracy, the simulated commands of every size were each tested for five loops, where each loop distributed the command 1 million times. The averaged results for the distribution speed and network bandwidth occupancy are shown in Figures 8 and 9, for the existing software and new system, respectively.

Table 1 Custom-defined Commands and Status Messages Based on RTS2 Protocol

Format	Explanation	Sender	Receiver
GV <key>	Get parameters	LAMOST-CCD	CCD-Master
XV <key> <value>	Set parameters	LAMOST-CCD	CCD-Master
EX	Exposure start	LAMOST-CCD	CCD-Master
SX	Exposure stop	LAMOST-CCD	CCD-Master
RD	Readout	LAMOST-CCD	CCD-Master
A <Message>	Warning information	CCD-Master	LAMOST-CCD
+ <Message>	Success information	CCD-Master	LAMOST-CCD
- <Message>	Failed information	CCD-Master	LAMOST-CCD
······	Other control command	LAMOST-CCD	CCD-Master

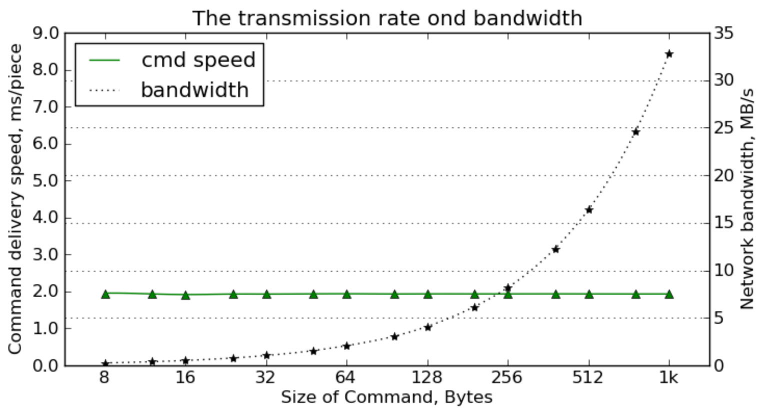


Fig. 8 Command distribution performance of the existing software, based on transmission rate and bandwidth as functions of command size.

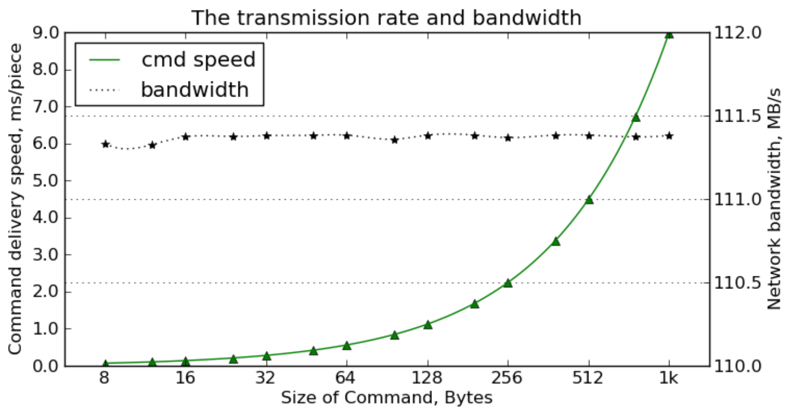


Fig. 9 Command distribution performance of the new system, based on transmission rate and bandwidth as functions of command size.

In the existing UDP-based software, the command distribution speed is constant at 1.94 ms/piece, and the network occupancy increases with the command packet size. However, in the new TCP-based system, because multi-thread technology is employed, the network occupancy is always close to the ultimate transmission speed (the gigabit switch's ultimate speed is 1 billion bps / 8 $\approx$ 119.21 MBps), and the command delivery speed increases with the command pack size. From comparison of the two figures, it is apparent that the new system exhibits superior performance with respect to the existing system when the command pack size is less than 200 B. In contrast, when the command pack size exceeds 200 B, the new system exhibits inferior performance with respect to the existing software. However, even when the command size is 1 KB, the delivery speed of the new system is as high as 8.94 ms/piece, which is far less than 100 ms, the UCAM time sensitivity. Therefore, the command distribution speeds of both software systems can satisfy the actual engineering requirements for LAMOST. In real observation scenarios, our camera commands are usually smaller than 200 B; thus, use of TCP is preferable to UDP. Further, using TCP, there is no risk of packet loss or out-of-order packets. Moreover, TCP provides more options for the already mature library implementation, which is expected to reduce the maintenance and redevelopment workload significantly. Considering all the above, we can conclude that the new system can more effectively satisfy the actual engineering requirements for LAMOST.

4.2 Status Transmission Performance

In the second experiment, we tested the status collection performance of the new system to determine whether it satisfies the engineering requirements for collecting and processing the status information of the 32 real CCD cameras. Status data packets (8 B, 12 B, ..., 1 KB) were created for each of the 32 cameras. Then, all these status packets were sent to our new system simultaneously. As previously, to guarantee the experimental accuracy, the simulated status packets of each size were tested for five loops, and each loop involved one million distributions. The averaged results for the transmission speed and occupancy network bandwidth are shown in Figure 10.

For simple development and maintenance, we only designed a single thread and one endpoint to receive and process all status packets. For the command experiment, from Figure 10, the status processing speed increases with the status packet size. However, the network occupancy is not very high and exhibits a peak at a status packet size close to 48 B. The reason for this phenomenon is that the software is limited by input/output (IO) blocking and sender judgment logic. However, even when the status packet size is 1 KB, the status receiving and processing speed is only 3.2 ms/piece. Because it is not critical for the status processing requirements to be satisfied in real observation scenarios, the single-camera status generation rate is approximately second level. Thus, the status processing performance of our developed system is adequate.

For details on reception performance and packet loss rate of the existing system, please see Dong et al. (2009). As this information has already been published, it is not described again here.

4.3 Automatic Observation

We wish to introduce RTS2 to the LAMOST OCS so as to implement its automatic telescope control capability and robustness, thereby overcoming the shortcomings of the existing camera-control software. Therefore, a third experiment was designed to test the automation and robustness of the new system.

RTS2 provides an observation decision-maker named “selector” (SEL), which can automatically select targets from a database according to various strategies (Kubánek et al. 2012a). The targets in database can be inserted or updated by RTS2scripts, e.g. rts2-newtarget or rts2-target. RTS2 also provides a device scheduler named “executor” (EXEC). These two core modules, along with the CentralD status synchronizer, guarantee the automation capability of RTS2. In addition, all RTS2 modules have pluggable implementation. Thus, the modules run independently and are connected dynamically. This property provides RTS2 with fault tolerance and robustness. It is possible to remove and add devices during observation without affecting the entire observation. Finally, RTS2 also provides a large number of dummy device modules for various types of astronomical equipment. The relationship between RTS2 modules for automatic observation is shown in Figure 11. Therefore, when our camera control software was realized, we could register our new system in the RTS2 environment and immediately begin simulated observations using other dummy device modules and servers.

We performed multiple observation tests continuously during a LAMOST maintenance session, using a calibration lamp instead of real observation targets. A three-loop observation was performed for each target, and a custom script was run for each observation to yield three repeated exposures of duration 5 seconds.

Figure 12 shows the procedure for one of these tests. In the observation experiment, we created a target table in a PostgreSQL database and ran the RTS2 automatic mode. Then, RTS2 SEL selected a suitable target from the database and sent the target identifier to EXEC. EXEC then drove the telescope (T0 in Fig. 12), which was an RTS2 dummy device module in this case. Our new camera-control system (C0 in Fig. 12) worked with EXEC to complete the observation task.

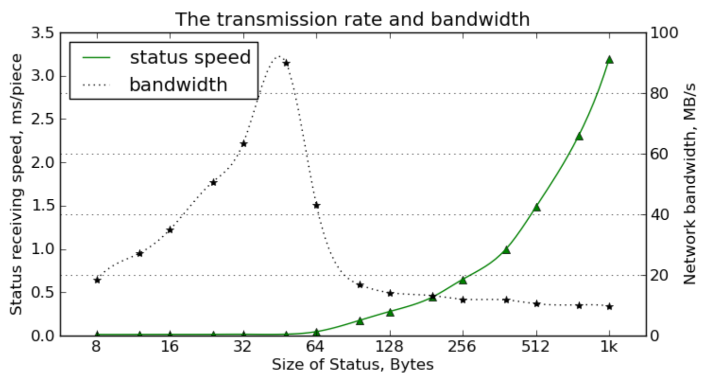


Fig. 10 Status collection performance of the new system.

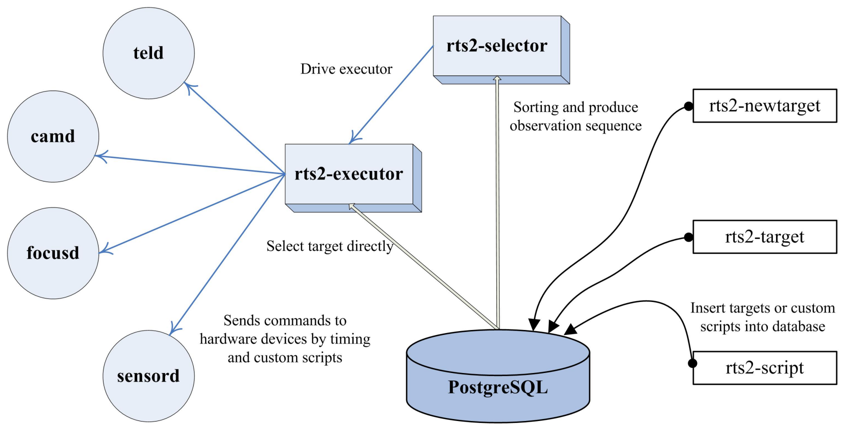
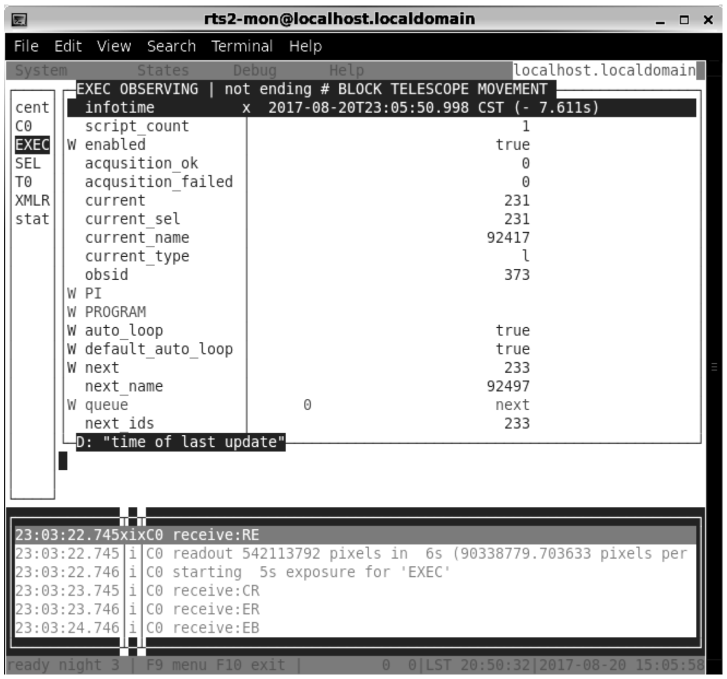

teld
camd
focusd
sensord
rts2-selector
rts2-executor
PostgreSQL
rts2-newtarget
rts2-target
rts2-script
Drive executor
Sorting and produce observation sequence
Select target directly
Sends commands to hardware devices by timing and custom scripts
Insert targets or custom scripts into database

Fig. 11 The schematic diagram of RTS2 automatic observation.


```
rts2-mon@localhost.localdomain
File  Edit  View  Search  Terminal  Help
System        States     Debug      Help              localhost.localdomain
      EXEC OBSERVING | not ending # BLOCK TELESCOPE MOVEMENT
cent    infotime              x  2017-08-20T23:05:50.998 CST (- 7.611s)
C0      script_count                                   1
EXEC  W enabled                                     true
SEL     acqusition_ok                                  0
T0      acqusition_failed                              0
XMLR    current                                      231
stat    current_sel                                  231
        current_name                               92417
        current_type                                   l
        obsid                                        373
      W PI
      W PROGRAM
      W auto_loop                                   true
      W default_auto_loop                           true
      W next                                         233
        next_name                                  92497
      W queue                    0                  next
        next_ids                                     233
      D: "time of last update"

23:03:22.745xixC0 receive:RE
23:03:22.745|i|C0 readout 542113792 pixels in  6s (90338779.703633 pixels per
23:03:22.746|i|C0 starting  5s exposure for 'EXEC'
23:03:23.745|i|C0 receive:CR
23:03:23.746|i|C0 receive:ER
23:03:24.746|i|C0 receive:EB
ready night 3 | F9 menu F10 exit |          0   0|LST 20:50:32|2017-08-20 15:05:58
```

Fig. 12 Observation using new camera-control software with RTS2 environment.

Figure 13 is a screenshot of the GUI of the new system for a given observation. This GUI can provide more detailed information on the statuses of the 32 EEV CCD cameras during operation than the existing system.

Figure 14 shows the results of a selected observation. Note that, after one observation, we obtain 32 target Flexible Image Transport System (FITS) files (Arc FITS files, to be specific).

The results of the observation experiment indicate that the new RTS2-based camera-control software runs automatically and stably, and satisfies the observation requirements for LAMOST.

5 DISCUSSION

5.1 Virtual Camera or 32 Camera Modules Added to RTS2 Framework

To create the customized RTS2 device module for LAMOST, two design schemes were considered, as discussed below.

5.1.1 First scheme

According to this scheme, we would create 32 of our own camera modules, all of which would inherit from the RTS2::Camera class and overwrite certain interfaces (virtual functions). The number of customized devices would simply be increased. This process appears simple; however, there are many hidden disadvantages and risks. RTS2 was originally designed for small telescopes, and the intermodule communication mechanism uses a TCP-based network structure topology (not a star structure topology). When we register new modules in RTS2, the number of intermodule connections increases geometrically. An excessive number of connections not only reduces the stability of the RTS2 system, but also occupies additional bandwidth (many misuse messages are transferred between the new camera modules).

5.1.2 Second scheme

According to this scheme, a virtual layer that bridges RTS2 and the 32 real cameras is added. In this scheme, we create a customized camera and register it in the RTS2 framework like a simple camera. Then, we add a virtual layer (proxy) responsible for command distribution, coordination of the 32 real cameras and status collection. This virtual layer simply provides an aggregate message to the RTS2 customized camera as feedback. By adding the virtual layer, we simplified the development process and reduced the coupling between the RTS2 and 32 real cameras. Another benefit also exists, i.e., we can reconstruct and upgrade the existing LAMOST CCD control software to reduce the workload and avoid development duplication. Based on the above (and the previous sections), it is apparent that we chose the second scheme for implementation of the LAMOST camera-control system.

5.2 Use of TCP instead of UDP

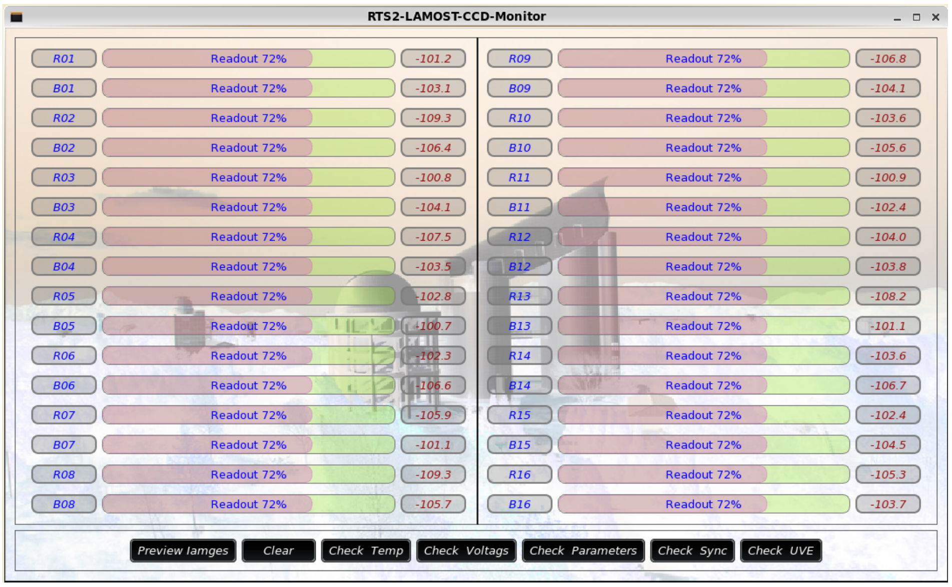
In the existing camera-control software, a UDP broadcast mechanism is used, with the expectation that commands can arrive at each camera simultaneously. In order to improve the UDP reliability, many auxiliary codes have been added. However, packet loss and out-of-order problems always occur (Dong et al. 2009). This problem is particularly serious in real observation scenarios, especially for cases involving large information throughput or a heavy network load. In addition, these auxiliary codes render the software obscure and increase the difficulty in maintenance.

On the other hand, TCP provides reliable, ordered and error-checked delivery of a stream of octets. This approach can overcome the shortcomings of UDP unreliability in the LAMOST high-speed local area network. From a technical perspective, TCP has lower efficiency than UDP. However, experiments have shown that careful TCP design can yield efficiency close to that for UDP. This is completely in line with the engineering requirements of LAMOST (see Sect. 4.1 and Sect. 4.2). Further, use of TCP as the transport protocol can render the software both simple and stable. Moreover, use of TCP simplifies integration of the RTS2 framework into the new LAMOST camera-control system.

In summary, in the new RTS2-based LAMOST camera-control system, the UDP-based communication mechanism is replaced with a TCP transport protocol.

5.3 Fault Tolerance and Exception Handling Mechanism

As an engineering project, the fault tolerance and exception handling mechanism is very important. It ensures stable operation of the entire software system and provides a mechanism for human-computer interaction if necessary. The fault tolerance and exception handling mechanism of the new RTS2-based LAMOST camera-control system is described in the following aspects:



R01	Readout 72%	-101.2	R09	Readout 72%	-106.8
B01	Readout 72%	-103.1	B09	Readout 72%	-104.1
R02	Readout 72%	-109.3	R10	Readout 72%	-103.6
B02	Readout 72%	-106.4	B10	Readout 72%	-105.6
R03	Readout 72%	-100.8	R11	Readout 72%	-100.9
B03	Readout 72%	-104.1	B11	Readout 72%	-102.4
R04	Readout 72%	-107.5	R12	Readout 72%	-104.0
B04	Readout 72%	-103.5	B12	Readout 72%	-103.8
R05	Readout 72%	-102.8	R13	Readout 72%	-108.2
B05	Readout 72%	-100.7	B13	Readout 72%	-101.1
R06	Readout 72%	-102.3	R14	Readout 72%	-103.6
B06	Readout 72%	-106.6	B14	Readout 72%	-106.7
R07	Readout 72%	-105.9	R15	Readout 72%	-102.4
B07	Readout 72%	-101.1	B15	Readout 72%	-104.5
R08	Readout 72%	-109.3	R16	Readout 72%	-105.3
B08	Readout 72%	-105.7	B16	Readout 72%	-103.7

Fig. 13 GUI of the new camera-control software during observation.

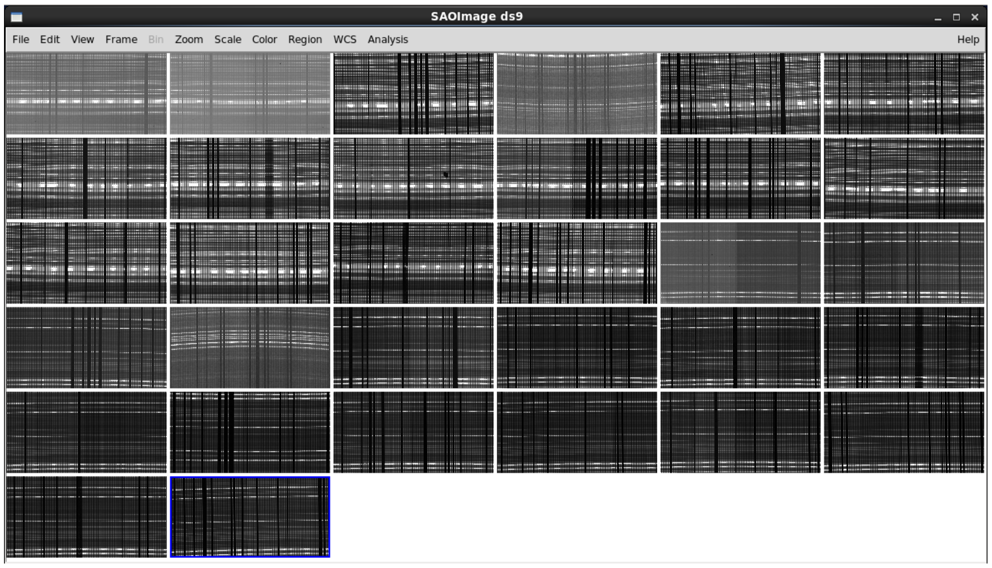


Fig. 14 Results provided by the new system for one observation.

Firstly, as a device module in the RTS2 framework, the new system inherits the fault tolerance of RTS2 directly. All RTS2 programs are designed to be fault tolerant. The failure of one device does not affect other devices executing daemons (Kubánek et al. 2006). It is possible to remove and add our virtual camera module during observation without affecting the whole observation.

Secondly, in CCD-Master, the status processor allocates a timer for each CCD-Slave to regularly track the key operation steps. For example, in a reading process, each CCD-Slave transfers a message about the readout progress (generated by UCAM) every second. The timers monitor these messages and create warning information if one CCD-Slave does not update the message within the specified time. The warning information will be broadcast to other RTS2 modules via RTS2's message mechanism (logStream() function). It also can trigger a GUI to pop-up a message box and remind the observer.

Thirdly, following the RTS2 connections paradigms, the connection between CCD-Slave and CCD-Master allows reconnection. When a single CCD camera encounters serious hardware problems, an engineer can close

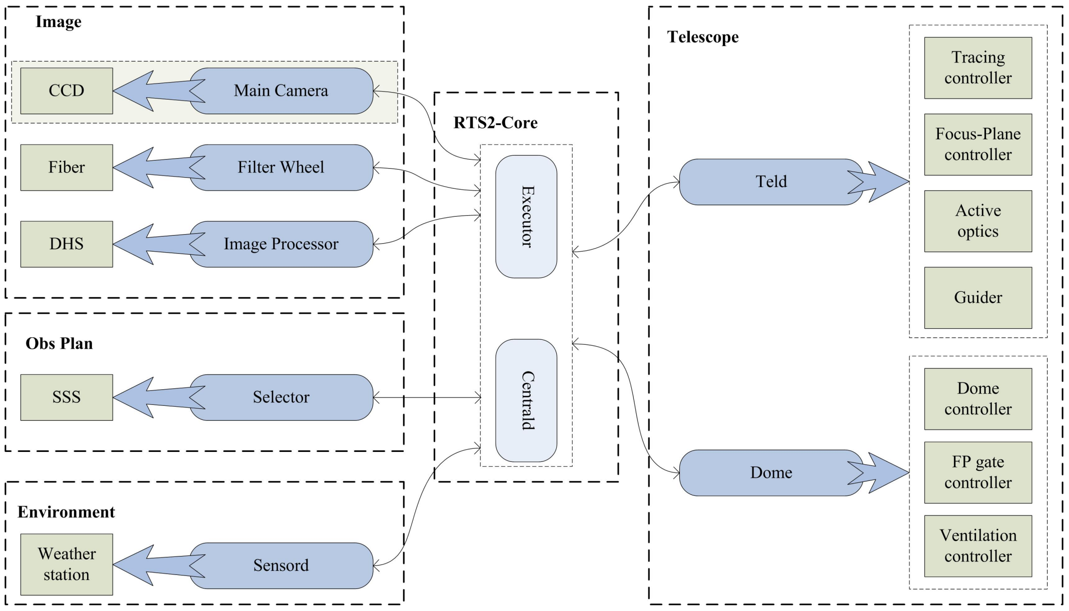


Fig. 15 Architecture diagram for extension of RTS2 to the entire LAMOST OCS.

this CCD-Slave and remove the connection. After the problem is solved, CCD-Slave can be started and the connection will be re-established automatically. This process does not affect other normal CCD operations.

The above considerations of fault tolerance and exception handling measures provide the necessary guarantees for stable operation of the new camera-control system.

5.4 RTS2 Extension to Entire LAMOST OCS

Virtualization of complex devices is a very innovative concept for upgrading the control software of a large-scale astronomical telescope. This approach can add features of the new software framework while retaining interfaces of the existing software with a minimum of possible changes. Hence, the software design is simplified and bugs caused by adding too many new codes are reduced, with development being accelerated.

In order to improve LAMOST's automatic observation capability, we introduced the RTS2 framework to the LAMOST camera-control software. The 32 CCD cameras installed in LAMOST were mapped into one RTS2 virtual camera module. The results of the validation experiments conducted in this study demonstrate the automatic observation capability of the new software.

The LAMOST OCS is an extremely large and complex software system, with the camera-control system being only one subsystem. To realize automatic observation capability for LAMOST, we must reconstruct other OCS subsystems, i.e., the tracking controller, fiber-positioning controller, etc. The architecture diagram for extension of RTS2 to the entire LAMOST OCS is shown in Figure 15.

Using the device virtualization approach, we will create additional virtual device modules to communicate with other existing OCS subsystems. Finally, by extending the RTS2 framework to the entire OCS, we will realize the ultimate goal of using RTS2 to control LAMOST.

Currently, modular design and layered implementation is the general trend for software development, and a large number of excellent new software frameworks are being used to implement this concept. To exploit the advanced features provided by these frameworks, compatibility issues between them and existing software must be addressed. The concept of existing device or software virtualization is an important method for resolving this problem. We believe that this concept is the guiding principle for introduction of the RTS2 framework to LAMOST and enhancing its automatic observation capability. In addition, this concept and its implementation will provide a reference for other large astronomical telescope software upgrades.

6 CONCLUSIONS

In this study, we designed and implemented a new camera-control system based on RTS2 for LAMOST. The results of a series of experiments and observation tests demonstrate the automation and stability of the new system, and indicate that the new system satisfies the observation requirements for LAMOST. In developing the new system, we reconstructed the existing software and implemented the concept of a virtual layer. These technologies greatly simplified the software development. Further, the approach described in this study will also provide a reference for software upgrades of other large-aperture astronomical telescopes, with the aim of realizing automation.

The camera-control system is only one part of the LAMOST OCS. In the future, other existing software packages (the mount, guider, online data handler, etc.) will be reconstructed for adaptation to the RTS2 framework. By retaining the concept of device virtualization, we are certain this task will be accomplished.

Acknowledgements This study is supported by the National Key Research and Development Program of China (Grant No. 2016YFE0100300), the Joint Research Fund in Astronomy (Grant Nos. U1531132, U1631129 and U1231205) under cooperative agreement between the National Natural Science Foundation of China (NSFC) and the Chinese Academy of Sciences (CAS), and the National Natural Science Foundation of China (Grant Nos. 11603044, 11703044, 11503042, 11403009 and 11463003). The Guo Shou Jing Telescope (the Large Sky Area Multi-Object Fiber Spectroscopic Telescope, LAMOST) is a National Major Scientific Project built by the Chinese Academy of Sciences. Funding for the project has been provided by the National Development and Reform Commission. LAMOST is operated and managed by National Astronomical Observatories, Chinese Academy of Sciences. We also thank the reviewers for helpful comments and suggestions that improved the paper.

References

Castro-Tirado, A. J. 2010, Advances in Astronomy, 2010, 570489

Cui, X.-Q., Zhao, Y.-H., Chu, Y.-Q., et al. 2012, RAA (Research in Astronomy and Astrophysics), 12, 1197

Cui, C., Li, J., Cai, X., et al. 2013, Progress in Astronomy, 31, 141

Deng, X., Wang, J., Dong, J., et al. 2010, in Proc. SPIE, 7740, Software and Cyberinfrastructure for Astronomy, 774036

Deng, X., Wang, J., Dong, J., et al. 2011, Astronomical Research & Technology, 8, 279

Dong, J., Wang, J., Deng, X. C., et al. 2009, Nuclear Electronics & Detection Technology, 29, 369

Hou, Y., Zhu, Y., Hu, Z., Wang, L., & Wang, J. 2010, in Proc. SPIE, 7735, Ground-based and Airborne Instrumentation for Astronomy III, 77350C

Jia, L., Wei, M., Zou, S., & Luo, Y. 2010, in Proc. SPIE, 7733, Ground-based and Airborne Telescopes III, 77335E

Kubánek, P., Jelínek, M., Vítek, S., et al. 2006, in Proc. SPIE, 6274, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, 62741V

Kubánek, P., Jelínek, M., French, J., et al. 2008, in Proc. SPIE, 7019, Advanced Software and Control for Astronomy II, 70192S

Kubánek, P., Falco, E., Jelínek, M., et al. 2012a, in Proc. SPIE, 8448, Observatory Operations: Strategies, Processes, and Systems IV, 844811

Kubánek, P., Prouza, M., Kotov, I., et al. 2012b, in Proc. SPIE, 8451, Software and Cyberinfrastructure for Astronomy II, 84512T

Wei, S., Chen, Y., Liang, B., et al. 2014, Astronomical Research & Technology, 11, 281

Zhang, G.-Y., Wang, J., Tang, P.-y., et al. 2016, MNRAS, 455, 1654

Zhao, Y. 2015, IAU General Assembly, 22, 2255903

Zou, S., & Wang, G. 2006, in Proc. SPIE, 6269, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, 626922