A control system for the ASTE Polarimeter: design and testing

Lühr Sierra, Daniel Vicente, Melillanca, Javier, Vargas, Felipe, Li, Hua-bai

Daniel Vicente Lühr Sierra, Javier Melillanca, Felipe Vargas, Hua-bai Li, "A control system for the ASTE Polarimeter: design and testing," Proc. SPIE 11452, Software and Cyberinfrastructure for Astronomy VI, 1145237 (13 December 2020); doi: 10.1117/12.2563012

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2020, Online Only

A control system for the ASTE Polarimeter: Design and testing

Daniel Vicente Lühr Sierra[a], Javier Melillanca[a,b], Felipe Vargas[a,b], and Hua-bai Li[c]

[a]Universidad Austral de Chile, Facultad de Ciencias de la Ingeniería, Valdivia, Chile
[b]Universidad de Concepción, Facultad de Ingeniería, Concepción, Chile
[c]The Chinese University of Hong Kong, Shatin, New Territory, Hong Kong, China

ABSTRACT

The ASTE Polarimeter (APol), developed by Dr. Li at the Chinese University of Hong Kong (CUHK), presented a simple but innovative approach to carry out polarimetric measurements using ASTE Telescope's TES camera. Our group at Universidad Austral de Chile (UACh) has collaborated in the project since its early stages and was assigned with the task of developing the control software for the instrument. The software has been developed also keeping the simplicity concept in mind. All its functionality has been separated in simple modules which are in charge of well defined tasks. The interfaces between the modules follow the design of modern applications and are based on well defined standards, such as those used by internet applications. The instrument has also the opportunity to be tested on the JCMT Telescope, and it is going to be used as the base design for a polarimeter in the future Leighton Chajnantor Telescope (LCT). Therefore, there is a requirement that the control software should be flexible enough to interface with at least these three telescopes, all of which run very different control software systems. This paper presents the design and implementation of APol's control software, as well as some results of laboratory tests of the instrument.

Keywords: Software development, Telescopes, Control software, Polarimeter, Interfaces

1. INTRODUCTION

1.1 APol: The ASTE Polarimeter

APol is a polarimeter instrument built originally to be installed at the Atacama Submillimeter Telescope Experiment (ASTE). It has been developed as a joint international collaboration leaded by Dr. Li Hua-bai and his team at the Chinese University of Hong Kong. The polarimeter is a fore-optics module to the ASTE's 350 GHz Transition Edge Sensor (TES) camera. APol's optical design and first lab-test results is described in 1.

APol consists of a rotating half-wave plate (HWP) mounted on a torque motor (TM) and a fixed wire grid (WG) as depicted in Fig. 1 and a control unit as an interface to the telescope system as well as controlling the TM and local storage of the instrument's data.

This paper details the software running in the control unit. The following sections will briefly introduce the hardware and software designs of APol's control unit.

1.1.1 APol's control unit hardware

APol's control unit purposes are to control the rotation of the TM and consequently the HWP, to log the angular position as well as other parameters and variables, and to communicate to the telescope's system.

Tha control unit is built around a Technologic Systems TS-8550-4900 embedded ARM computer. The motor is controlled by a Copley Controls Xenus XTL-230 servo controller with CANopen interface. The TS-8550-4900 computer includes a CAN port which is used to communicate with the Xenus controller.

APol's data is stored locally in a solid state disk connected to the embedded computer's SATA II port.

Further author information: (Send correspondence to Daniel Vicente Lühr Sierra)
Daniel Vicente Lühr Sierra.: E-mail: danielluhr@uach.cl, Telephone: +56 63 222 1868
Hua-bai Li: E-mail: hbli@cuhk.edu.hk

Software and Cyberinfrastructure for Astronomy VI, edited by Juan C. Guzman, Jorge Ibsen, Proc. of SPIE
Vol. 11452, 1145237 · © 2020 SPIE · CCC code: 0277-786X/20/$21 · doi: 10.1117/12.2563012

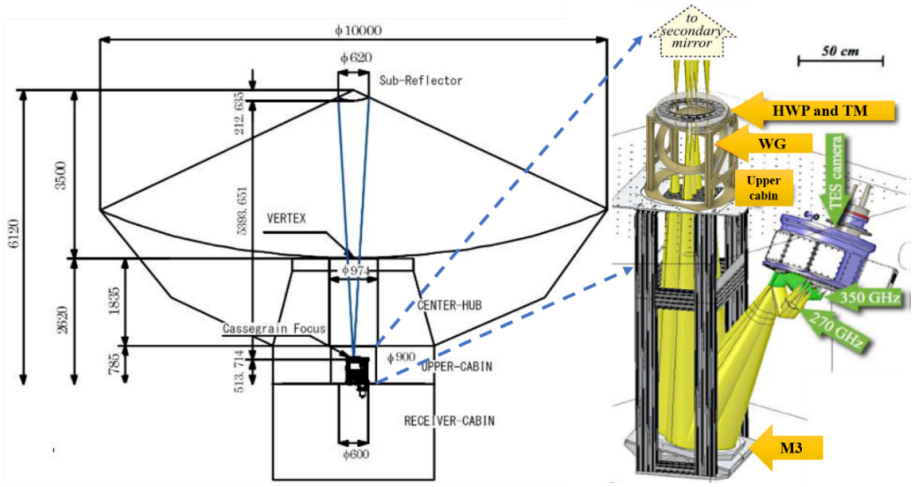

Figure 1. Schematic drawing of APol in ASTE. APol optical components are installed in ASTE's upper cabin in the optical path between the secondary and the tertiary mirrors.[1]

In order to simplify network connections to the control unit, either to interface to the telescope's network or to directly connect a terminal for configuration, monitoring and debugging, the control units has an Ubiquity EdgeRouter-X Gigabit Ethernet network router.

The control unit also has power supplies for all the electronic components (24Vdc, 12Vdc and 5Vdc buses), thermomagnetic switches, and emergency stop switch, and can optionally include a small DC-UPS.

A block diagram of the main components is depicted in Fig. 2 (left). The Figure (right) also shows a picture of the actual box containing the control unit (SSD and Router not shown).

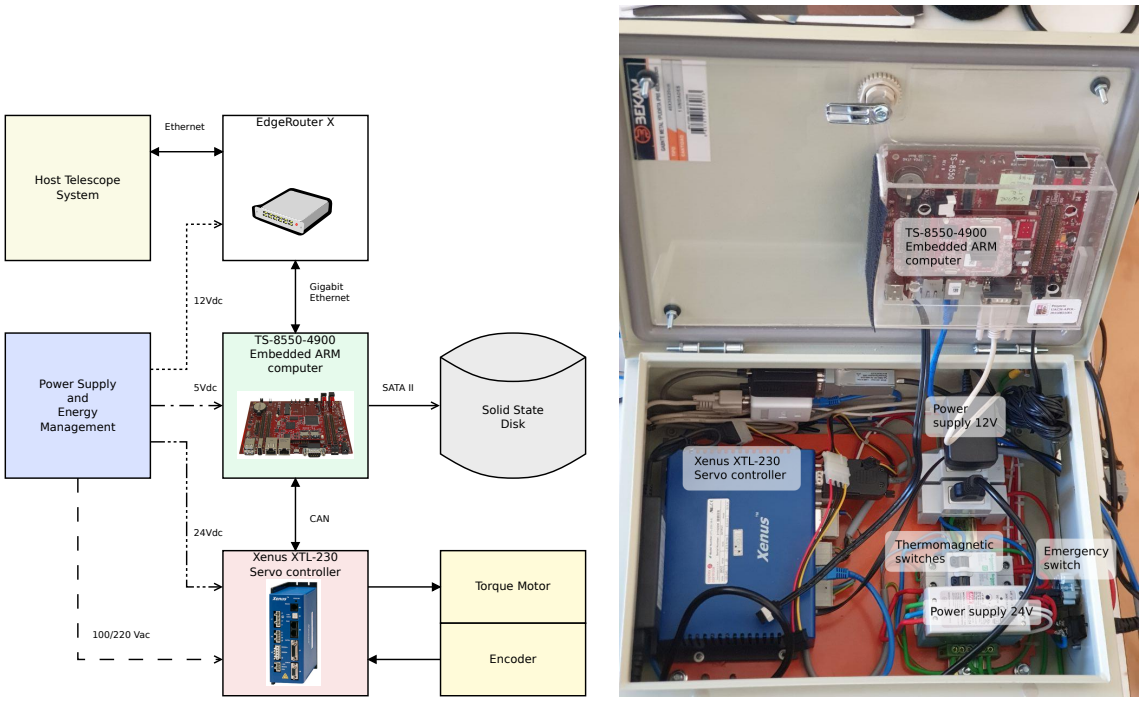

Figure 2. Left: Block diagram of APol's control unit. Right: APol's control unit box.

Although originally developed to be installed at ASTE, there is also an opportunity to carry out preliminary tests at JCMT. Therefore, the hardware, as well as the software, has been designed to be as flexible as possible in order to adapt to the different control systems of diverse telescopes.

1.1.2 APol's control unit software

The TS-8550-4900 embedded ARM computer runs GNU/Linux as its operating system. Technologic Systems provide software utilities and programming libraries to configure and access all specific features of the board. A python module implementing a library to communicate with the Xenus XTL-230 servo controller using the CANopen protocol was developed. A REST server was implemented to provide the core functionality of APol. The server can be accessed directly through API calls or lightweight clients can be developed according to the specific requirements of the host telescope's control system. The details of the software are presented in section 2.

1.2 Other alternatives ROS and ACS

Although the current control software was designed from the bottom-up specifically for the instrument and the telescopes which will host it, another approach would have been to use a standard framework.

A framework is a set of classes that incorporate an abstract design to solve a family of related problems2. Generally, it consists of a mixture of abstract and concrete classes, which are also known as class libraries.

A framework approach improves the flexibility and generally leads to a modular design. However, it usually involves some extra overhead since frameworks are very general and even in their minimal configuration they provide features which might not be needed for our instrument at this moment. Nevertheless, aiming at a long term development of the polarimeter control software which could work with newer versions of the polarimeter or a wider range of telescopes the use of an appropriate framework is advisable. Since the the developed software has already been developed in a modular and lightweight approach, incorporating it into an existing framework should not be a difficult process.

Next, we will comment two frameworks which we have considered as alternatives for future work.

1.2.1 ROS

ROS (Robot Operating System) is an open source project and an operating meta-system for robots, delivering services such as, hardware abstraction,[3] low-level device control, functionalities implementation, message transfer between processes and package management, among others.

ROS is highly widespread and also has a fairly large development community to provide solutions to specific problems of the framework. Although, ROS is a framework developed for use in robotics, it has been used in other automation applications in industry. In particular, it has been applied to telescope control. Experiences carried out "Ckoirama" observatory of the University of Antofagasta, located 90km from the city of Antofagasta,[4] and ICETel belonging to CSIC,[5] close to the city of Barcelona demonstrate such specific application.

1.2.2 ACS

The ALMA Common Software (ACS) is an application framework designed to provide a common and homogeneous software architecture and infrastructure, spanning the end to end needs of an Astronomical observatory, from the Telescope Control system to high-level data flow management.[6] ACS is based in CORBA LGPL public license and since the beginning it has been developed as a general purpose application framework (i.e. not specifically linked to ALMA needs).

2. APOL CONTROL: FROM CONCEPT DESIGN TO IMPLEMENTATION

One of the main tasks of APol's software is to control the TM where the HWP is located and to record its angular position. This is all done through the Xenus XTL-230 servo controller. In order to achieve the maximum speed when transferring commands and data, the Xenus XTL-230's CANopen interface is used, instead of the RS-232 one. The Xenus XTL-230 complies with CAN in Automation (CiA) 402 standard. A python module to access most of the Xenus XTL-230 controller functions implemented by the CiA-402 over the CANopen protocol was implemented. It is described in section 2.1.

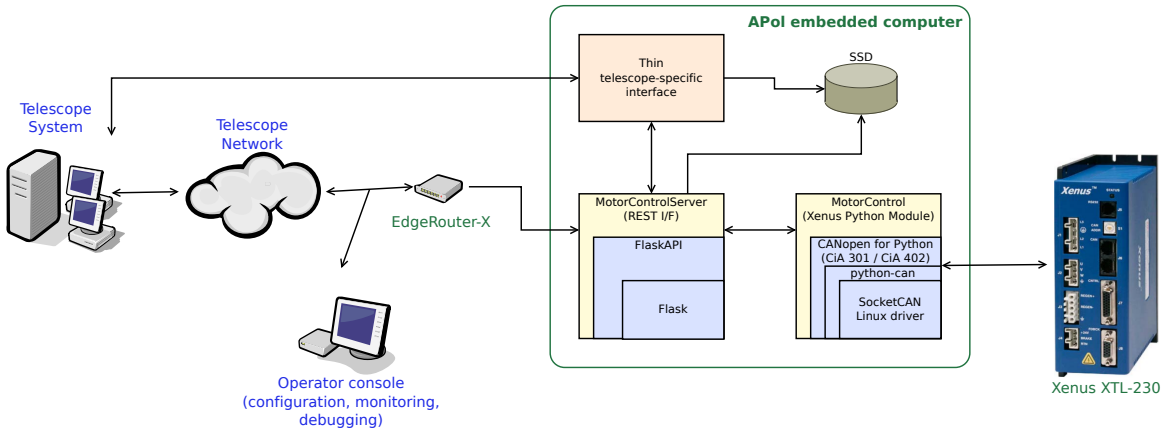

Figure 3. APol's control unit software block diagram.

The basic high-level logic of APol is provided through a REST API served by a Flask-based application. With this approach the interface "glue" between the telescope's system and APol can be developed in any programming language. The details of this REST server are presented in section 2.2. The telescope's system can interface directly to the Flask-based server application if there is properly configured network. Otherwise, a "thin" interface specifically built for a telescope can be implemented in APol's embedded computer and communicate with the telescope's system using their available hardware and software interfaces.

The REST server application also implements the local data recording, as detailed in section 2.3. It collects variables and parameters from the controller as well as from the embedded computer and records them to disk periodically, adding a timestamp to each data entry. Therefore, it is critical that APol's embedded computer clock is in sync with the telescope's main clock, in order to fuse the data collected by other telescope's instrument with APol's data. This topic is discussed further in section 2.4.

A block diagram of the software components is presented in Fig. 3.

2.1 CAN based hardware interface

Controller Area Network (CAN) is one of the most widely used industrial communication protocols today, providing communication services between different devices connected on a single network, both locally and remotely. The high speed and reliability of data transport have transformed it into a widely used protocol for applications in highly integrated control systems.[7]

CANopen is an open protocol that facilitates the development of applications, due to its flexibility of interconnection of devices in the network, taking full advantage of the CAN bus, transforming it into a high-level protocol friendly for integrators and developers.[8]

The CANopen protocol is defined under sub-protocols structured in object dictionaries, each device with this standard has a profile with all the operating characteristics. All the control systems that operate with CANopen do so through a closed-loop, coordinated by a master device, which uses the network to transmit and receive the commands enabled in the object dictionary of the device to be controlled.

The communication between CANopen nodes, implemented for this project, is mainly defined by three types of messages, Service Data Object (SDO) used for the configuration and parameterization of the devices, Process data Objects (PDO) used during the normal operation of the network to transfer data in real-time and Network management functions that represent the profile of specific functionalities for the device such as manufacturer data.[9]

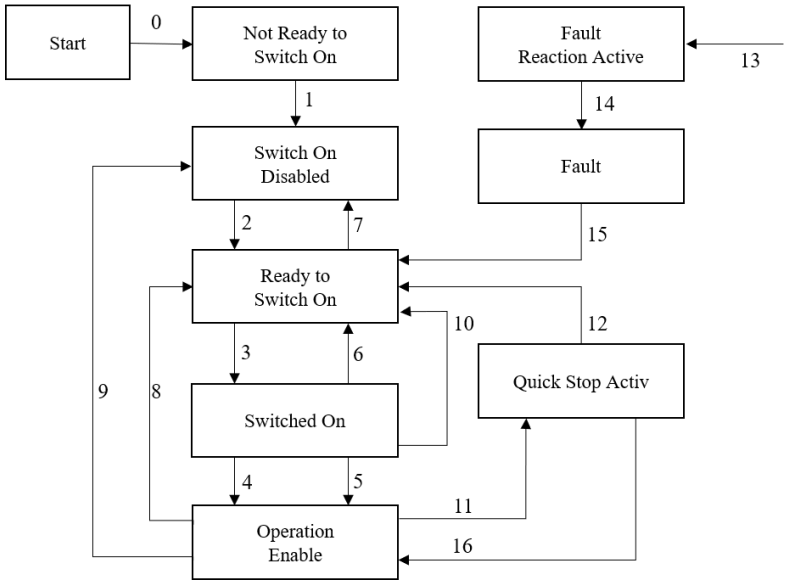

Figure 4. State Machine CiA 402 standard.

2.1.1 CiA 402

The CAN in Automation (CiA) 402 is a set of standardized specifications to control the behaviour of different devices. It is structured in such a way that each required profile is easily identifiable utilizing configurable parameters, whose states vary depending on the control word that is sent by the host. This makes CiA 402 widely used by different vendors who implement subsets of parameters of this standard. Each sub-assembly can contain one or more configuration profiles, as required for the application, such as position, speed or torque.[10] In the Fig. 4 the possible state change sequences of an amplifier are shown.

2.1.2 The Motor Control Python Module

The Motor Control Module is a class to communicate with Copley Controls Xenus Motor Controller via CANopen. This module contains the different commands to query the status of a variable or to configure the operating parameters of the controller. These commands providing access control the device, use the state machine of Fig. 4 and defines the behaviour of the device with the modes of operation. The table 1 shows a summary of the features implemented in the Python module. Not all the Xenus controller functions exposed through the CANopen interface have been implemented. Some of them are redundant and possible deprecated functions whose access is available through other SDO or PDO, while other have not been implemented yet because they are related to operation modes that are not required for the specific application. However, any feature not specifically implemented in the library can still be accessed through raw communication using the CANopen for Python module indicating the particular target SDO or PDO.

The Python module is built on top of the CANopen for Python module.[11] The class relationships are shown in the simplified UML class diagram of Fig. 5. The CANopen for Python module further depends on the python-can module[12] and the GNU/Linux's SocketCAN implementation developed by Volkswagen Research.[13]

Due to the complexities involved in the usage of the CiA 402 State Machine, a simplified interface to the library has been developed in the form of a REST API implemented over a web application server. This software component is described in the next section.

2.2 REST based software interface

As it was mentioned above, an abstraction layer to the Python module interacting with the hardware was implemented in the form of a REST server. REST or *Representational state transfer* was defined by Roy Fielding in 2000.[14] Before opting for a REST based approach, also XML-RPC (Remote Procedure Calling)

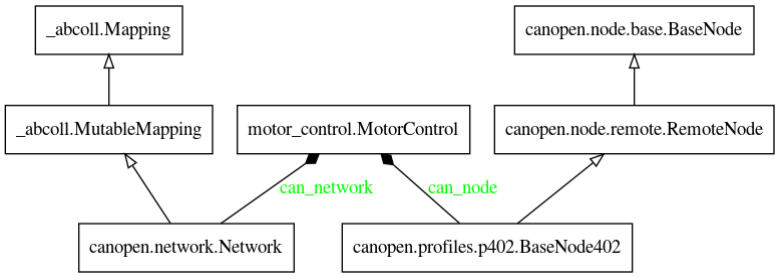

Figure 5. UML class diagrams for the APol's motor control library.

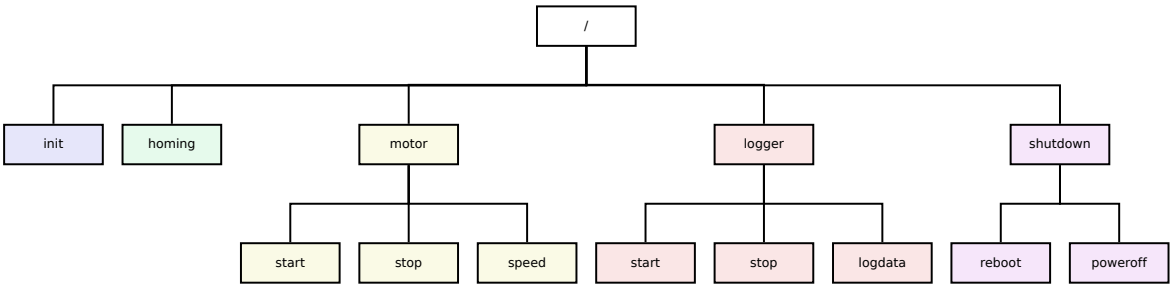

Figure 6. Tree structure of REST resources exposed by the server application.

protocol[15] was also considered. In fact, the first proof-of-concept of the system was based on XML-RPC. The REST architectural style was chosen because it was found to simplify the clients code and the availability of ready to use (out-of-the-box) implementations and frameworks.

The server is built on top of the Flask micro web framework written in Python.[16]

APol logic is exposed by the REST server as available *resources*. Currently the top-level resources are: `/init`, `/homing`, `/motor`, `/logger` and `/shutdown`. The `/motor`, `/logger` and `/shutdown` resources have second-level resources implemented as shown in Fig. 6

The root node of the tree (`/`) presents status information and the URLs to access the `/homing`, `/motor`, `/logger` or `/shutdown` resources if the Xenus controller has already been initialized, as reported by the servo controller itself. Otherwise it prints a warning message explaining the controller is not yet initialized and presents an URL to the `/init` resource which triggers the initialization process.

The `/homing` resource starts the homing operation of the controller which finds the zero position of the motor encoders. The progress of the operation is shown and after it finishes an URL back to the root node is presented.

By accessing to the `/motor` resource, a page with status information about the motor is shown, including actual and target velocities. Also, the URLs to access the resources under this node are presented: `/motor/start`,

Table 1. Summary code implemented Motor Control Module

Total lines of code	4739
Xenus functions implemented	58.57% (164 out of 280)
Functions detail	Network Management.
	Device control, configuration, and status.
	Control loop configuration.
	Stepper mode Support.
	Homing Mode
	Operation profiles: position, velocity and torque mode.

`/motor/stop`, `/motor/speed/`. The `start` and `stop` resources send the motor the instruction to reach the target velocity or set the velocity to zero, respectively. The `speed` resource is used to set the target speed in RPM.

The `/logger` page indicates the status of the data logging and the `/logger/start`, `/logger/stop` and `/logger/logdata` sub-resources. The first two starts and stops the data-logging facility respectively, while the third one actually writes an entry to the disk. This resource is called periodically by the data logging implementation described in section 2.3.

Finally, the `/shutdown` resource provide the `/shutdown/reboot` and `/shutdown/poweroff` URLs which tell the computer to cleanly (i.e. turn-off the Xenus controller properly first) reboot or power-off.

2.3 Data logging to disk

Since the REST-based server is accessing the Xenus XTL-230 controller at the highest data rate possible by the CANopen interface, it is very important to also record the angular position in timely manner. Also, it was decided that logging the data at regular intervals was desirable. Therefore, a data-logging class was implemented based on a Timer Object from Python's threading interface module.

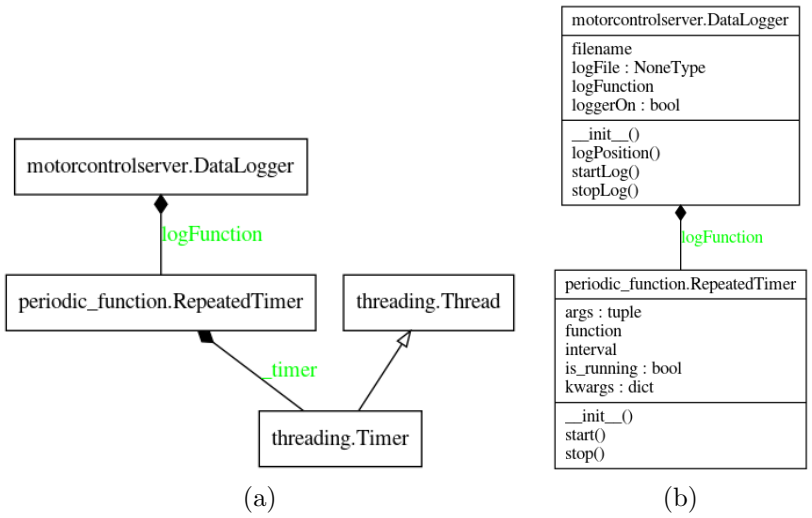

Figure 7. UML class diagrams for the APol's data-logging class. Class relationships are shown on the left (a). Main methods are shown on the right (b).

2.4 Time synchronisation

To achieve time synchronization with the telescope's system, APol control unit rely on a Network Time Protocol[17,18] (NTP) server. To keep drift to a minimum and if the signal is available, as in the case for ASTE, a *Pulse-per-second* (1PPS) signal can be processed by the embedded computer, using the included GNU/Linux kernel driver for such signals.[19]

3. CURRENT PROGRESS AND TEST RESULTS

APol's control software is currently being developed at the Universidad Austral de Chile (UACh) and is in the last phase of development before being integrated with APol's optics at the Chinese University of Hong Kong (CUHK) for lab tests and field tests after that. It is important to note that the development pace has been hindered by the global Covid-19 crisis. In particular, field tests at JCMT originally scheduled on the first quarter of 2020 have been postponed. Also, temporal lock-downs in the cities of both universities have imposed restrictions to access facilities and resources necessary for the development.

The control software is operational and is undergoing performance tests. Particularly, it is important to determine how fast the angular positions can be requested from the controller and written to disk without data losses or communication buffer overflows which might make the system to fail.

4. FUTURE WORK

As soon as current testing at the UACh is finished, the control unit box will be sent to the CUHK to integrate it with APol's optics. It will then be tested in the lab to check whether the complete system works according to the specifications. When the lab tests are complete, the instrument will be ready to be deployed and tested on the field to the earliest opportunity.

On the other hand, it was mentioned before that integration to an existing control framework used on telescopes such as ACS or ROS would be an important improvement of the software. This will be also be a future line of development. The current REST-based software architecture is the correct approach to ease the integration either with ACS or ROS since a just "thin" layer needs to be developed to interface the framework with the REST-based server.

Furthermore, the current version of the server uses Flask integrated web server, which is acceptable for tests and small prototypes but is not adequate for full production deployments. Therefore, the code will be update for a full web service integration. Among other benefits, it will permit the implementation of more complex standard authentication schemes such as HTTP-Basic, HTTP-Digest or HTTP-Token. It will also allow a reliable way to access the instrument remotely.

Finally, a new collaboration with Universidad de Concepción (UdeC), related to the Leighton Chajnantor Telescope (LCT), opens the possibility for the group to develop an improved polarimeter for the LCT based on the experience gathered while developing APol. The control software will thus need to be adapted to both the LCT system's interface and the new polarimeter hardware design.

ACKNOWLEDGMENTS

We would like to acknowledge the Research Grants Council of Hong Kong: General Research Fund (14307118, 14304616, 14600915, 14307019); and CASSACA-CCJRF (2016 Call) (Project ID 1607), for their support to the project.

REFERENCES

[1] Zhang, Y., Li, H.-B., Lühr, D., Takekoshi, T., Oshima, T., and Gu, Q., "Atacama sub-millimeter telescope experiment polarimeter (APol) I: design and lab-test result," *Appl. Opt.* **59**, 2593–2599 (Mar 2020).

[2] Srinivasan, S. et al., "Design patterns in object-oriented frameworks," *Computer* **32**, 24–32 (1999).

[3] Quigley, M. et al., "ROS: an open-source Robot Operating System," *IEEE International Conference on Robotics and Automation, ICRA* **3** (2009).

[4] Villalobos, A. et al., "Diseño e implementación de un observatorio robotico teleoperado basado en Robot Operating Systems," *Ingeniare - Revista Chilena de Ingenieria* **26**, 12–19 (2018).

[5] Vilardell, F. et al., "Using Robotic Operating System (ROS) to control autonomous observatories," *SPIE, Software and Cyberinfraestructure for astronomy IV* **9913** (2016).

[6] Dimarcantonio, P. et al., "An overview of the ALMA Common Software (ACS)," *Mem. S.A.lt* **78**, 719–722 (2007).

[7] Farsi, M. and Ratcliff, K., "An introduction to CANopen and CANopen communication issues," in [*IEE Colloquium on CANopen Implementation (Digest No. 1997/384)*], 2/1–2/6 (1997).

[8] Li, W., Lijin, G., and Zheyuan, L., "Design of stm32-based canopen motion control master in transfer robot," in [*2013 Third International Conference on Instrumentation, Measurement, Computer, Communication and Control*], 1609–1612, IEEE (2013).

[9] "Copley Motion Objects (CMO) Programmer's Guide 2012," (2012).

[10] Voss, W. and Hatfield, M., "Canopen–higher layer protocol based on controller area network (CAN) supports device profiles for i/o modules, motion control," (2005).

[11] Sandberg, C., "CANopen for Python." https://canopen.readthedocs.io/en/latest/ (2017).

[12] "python-can documentation." https://python-can.readthedocs.io/en/stable/ (2018).

[13] "SocketCAN GNU/Linux kernel driver." https://www.kernel.org/doc/Documentation/networking/can.txt (2017).

[14] Fielding, R. T., *Architectural Styles and the Design of Network-based Software Architectures*, PhD thesis, University of California, Irvine (2000).
[15] Winer, D., "XML-RPC Specification." http://xmlrpc.com/spec.md (Jun 1999).
[16] Mufid, M. R., Basofi, A., Al Rasyid, M. U. H., Rochimansyah, I. F., and Rokhim, A., "Design an mvc model using python for flask framework development," in [*2019 International Electronics Symposium (IES)*], 214–219 (2019).
[17] Mills, D., Martin (Ed.), J., Burbank, J., and Kasch, W., "Network Time Protocol Version 4: Protocol and Algorithms Specification." RFC 5905 (Proposed Standard) (June 2010). Updated by RFCs 7822, 8573.
[18] Mizrahi, T. and Mayer, D., "Network Time Protocol Version 4 (NTPv4) Extension Fields." RFC 7822 (Proposed Standard) (Mar. 2016).
[19] Mogul, J., Mills, D., Brittenson, J., Stone, J., and Windl, U., "Pulse-Per-Second API for UNIX-like Operating Systems, Version 1.0." RFC 2783 (Informational) (Mar. 2000).