Flight control software for the wave-front sensor of SUNRISE 1m Balloon Telescope

Alexander Bell[a] and Peter Barthol[b] and Thomas Berkefeld[a] and Bernhard Feger[a] and Achim M. Gandorfer[b] and Frank Heidecke[a] and Michael Knoelker[c] and Valentin M. Pillet[d] and Wolfgang Schmidt[a] and Michael Sigwarth[a] and Sami K. Solanki[b] and Dirk Soltau[a] and Alan M. Title[e]

[a]Kiepenheuer Institut für Sonnenphysik, 79100 Freiburg, Germany;
' [b]Max-Planck-Institut für Sonnensystemforschung, 37191 Katlenburg-Lindau, Germany ;
[c]National Center for Atmospheric Research, Boulder CO, USA;
[d]Instituto de Astrofisica de Canarias, La Laguna, Spain;
[e]Lockheed Martin Space Systems Co. Palo Alto, USA

ABSTRACT

This paper describes the flight control software of the wave-front correction system that flew on the 2009 science flight of the Sunrise balloon telescope. The software discussed here allowed fully automated operations of the wave-front sensor, communications with the adaptive optics sub-system, the pointing system, the instrument control unit and the main telescope controller. The software was developed using modern object oriented analysis and design techniques, and consists of roughly 13.000 lines of C++ code not counting code written for the on-board communication layer. The software operated error free during the 5.5 day flight.

Keywords: Balloon, Sun, Telescope, Object Oriented Programming

1. INTRODUCTION

The SUNRISE * balloon-borne telescope for solar observations performed its first science flight in June 2009 on a NASA long-duration balloon flight from Kiruna, Sweden to Somerset Island in North-East Canada.[1]

Stratospheric balloon-borne telescopes have two fundamental advantages over ground based telescopes: they permit UV observations and they provide a seeing-free, diffraction-limited image quality over the full field of view (FoV). However, pointing to the Sun and tracking a feature on the solar surface is a formidable task, especially for a telescope hanging on a balloon that is driven by stratospheric winds at an altitude of 36 km. In addition to the apparent (diurnal and seasonal) motion of the Sun, there is a number of oscillatory modes that may be induced by variable winds in the stratosphere, and taken up by the balloon-gondola system.

For the correction of low-order aberrations, a Correlating Wave-Front Sensor (CWS) was used. It consisted of a 6-element Shack-Hartmann wave-front sensor (WFS), a fast tip-tilt mirror for the compensation of image motion, and an active telescope secondary mirror for focus correction. The CWS delivered a stabilized image with a precision of 0.04 arcs (RMS), whenever the coarse pointing was better than $\pm$ 45 arcs peak-to-peak. The automatic focus adjustment maintained a focus stability of 0.01 waves in the focal plane of the CWS.

The next section describes briefly the design of Sunrise focussing on the computers involved in the CWS. The communication software used by the CWS is discussed in Section 3 while Section 4 deals with the implementation of an autonomous mode used during flight. Finally, Section 5 concludes this paper.

Software and Cyberinfrastructure for Astronomy, edited by Nicole M. Radziwill, Alan Bridger,
Proc. of SPIE Vol. 7740, 774003 · © 2010 SPIE · CCC code: 0277-786X/10/$18 · doi: 10.1117/12.857003

Figure 1. SUNRISE shortly after launch

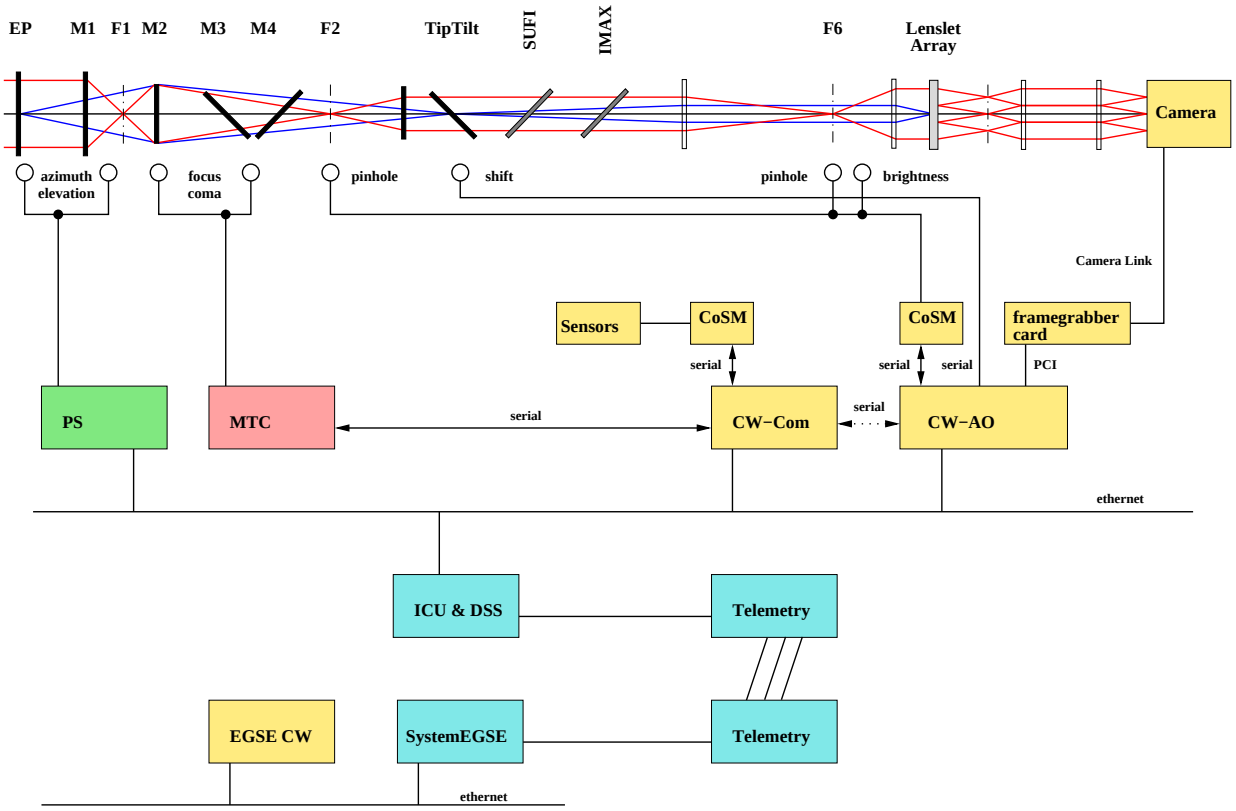

Figure 2. Scheme of the SUNRISE CWS hardware and communication lines. Hardware components that are built or contributed by KIS are shown as yellow boxes, PS denotes the gondola pointing system and MTC the main telescope controller.

2. SUNRISE OVERVIEW

An overview of the CWS architecture is given in Figure 2 where black lines are communication paths that are typically labeled with the interface used. The upper region shows the light path from the entrance pupil (EP) of the telescope to the camera. For a discussion of these components see.[2]

All communication between the CWS and other instruments are handled by the CW-Control computer (CW-Com), a compact A9M9750 module based on a NetSilicon, 200 MHz, NS9750 micro-controller dissipating less than 2 W in operating mode. CW-Com interfaces to:

CW-AO the computer doing the actual wave-front correction, a conduction cooled VME single board computer with two 1 GHz Motorola 7457 processors. Attached to CW-AO are the camera (using a PCI based frame grabber card), tip-tilt mirror (via two serial interfaces), mechanisms (via a serially connected CoSMbus[3]) controlling field stop, pinhole and dark in the F2 (telescope) and F6 (CWS) foci and a density filter that keeps the WFS camera illuminated appropriately. All communication with CW-AO is done via sockets using IP sockets over ethernet.

CoSM a serially connected bus used for collecting data from temperature sensors.

MTC the main telescope controller, delivered by the manufacturer of the telescope, Kayser Threde (Munich, Germany). A serial connection used for collecting house keeping information from the telescope as well as commanding the telescope. Note that all communication with the telescope must pass through CW-Com.

PS the pointing system provided by the High Altitude Observatory in Boulder (Colorado) using IP sockets. Commands in both direction are used here, CWS sends offloading commands to the PS to compensate sun rotation while tracking a target and the PS sends CWS so called "lock" commands whenever there is a good opportunity to lock the wave-front correction control loop.

ICU the Instrument Control Unit built by the Max Planck Institute for Solar System Research in Kattlenburg (Germany) via ethernet. Commands either originating from the ICU itself or from the CWS Electronic Ground Support Equipment (EGSE) are accepted from the ICU. In return housekeeping data is sent to the ICU.

A discussion of the specific communication and their realization is given in the following section.

3. COMMUNICATION SOFTWARE DESIGN

The flight control software was implemented in C++ using the Adaptive Communication Environment (ACE[4]), an open-source framework supporting patterns and principles required for the efficient development of concurrent networked applications. This lead to a multi threaded application running on CW-Com. For all incoming connections the reactor[5] framework is used. We will give an introduction to this framework and its application inside the flight control software in Section 3.1. Thereafter we discuss communication with the ICU in Section 3.2. Section 3.3 is dedicated to the communication with the MTC.

Further author information:
Alexander Bell: E-mail: albe@kis.uni-freiburg.de, Telephone: +49-761-3198323

*SUNRISE is an international collaboration of the Max-Planck Institut für Sonnensystemforschung (Katlenburg-Lindau, Germany), the High Altitude Observatory in Boulder (Colorado, U.S.), the Lockheed-Martin Solar and Astrophysics Laboratory (Palo Alto, U.S.), the IMaX Consortium (Tenerife, Granada, Madrid, Spain) and the Kiepenheuer-Institut für Sonnenphysik (Freiburg, Germany).

3.1 Handling incoming connections using the reactor framework

The reactor framework allows us to handle multiple I/O sources, in our case various socket connections as well as serial connections seemingly simultaneously within a single thread. For each connection where we have to react on an incoming event we derive a client class from `ACE_Event_Handler`, e.g., for a socket communication class:

```
class SocketClient : public ACE_Event_Handler
```

The client class is responsible to establish the connection, this is typically done by its constructor or by an "open" method. For the example client we have to start listening on a socket, represented by an `ACE_SOCK_Acceptor`.

```
ACE_SOCK_Acceptor mAcceptor;
mAcceptor.open(ServerAddr, 1)
```

Note that we skip error handling for clarity and brevity. The reactor accesses the handle of a client class by calling the method `get_handle`, which is easy to implement as long as the communication mode is supported by ACE:

```
ACE_HANDLE get_handle() const {
    return this->Acceptor.get_handle(); }
```

The most important method each reactor client must implement is

```
int handle_input(ACE_handle)
```

This method gets called by the reactor event loop whenever input arrives on the corresponding communication source. For a socket based communication the arriving data is read from the socket, analyzed and an appropriate answer is sent. Each client has to be registered at the reactor by calling

```
ACE_Reactor::instance()->register_handler()
```

which instructs the reactor to look for incoming events on the clients handle.

In the CW-Com software we use the reactor framework to process data received via a serial line from the MTC (see below) and to accept commands from the CW-AO computer and the pointing system an socket connections.

3.2 ICU communication

As each SUNRISE instrument CWS communicates with the ICU by receiving commands and sending housekeeping.

3.2.1 Receiving commands

Commands, whether originating from the EGSE or from the ICU are received using a pipe base communication protocol developed at the Max Planck Institute for Solar System Research using the ACE library. For a detailed discussion of this protocol see.[6] The CWS as a SUNRISE instruments receives low and high priority commands. For both of them a queue of commands is realized in the flight control software and the commands are executed on a first come first serve schedule where low priority commands are only executed when no high priority command is queued. In the flight control software this is realized inside the main thread by an event loop.

3.2.2 Housekeeping

CWS housekeeping is sent to the ICU every 5 seconds. We use a separate thread for this job triggered by a cyclic timer. Inside this thread we collect data from the temperature sensors connected to CW-Com, request status information from the CW-AO computer using a socket based communication.

3.3 MTC communication

The main telescope controller (MTC) is physically connected to the CW-Com using an RS422 interface, hence, all communication between SUNRISE and the telescope must pass CW-Com. All communication between the MTC and the ICU is forwarded by CW-Com transparently. Commands for the MTC either originating from the ICU, or an EGSE are realized as a special CW-Com command where the binary parameters of this CW-Com command correspond to an MTC command, i.e., all MTC commands are queued in the CW-Com command queue before they are passed on to the MTC. Housekeeping received from the MTC is packaged into CW-Com housekeeping packages and flagged with a special that allows the ICU to differentiate between the various MTC housekeeping packets and CW-Com housekeeping.

CW-Com itself communicates with the MTC directly when commanded to do so by the CW-AO during initialization (x, y, and z of the M2 mirror) and focus correction (z only). These commands are sent inside the `handle_input` method of the reactor client responsible for the communication with the CW-AO.

4. AUTOMATION

In order to minimize communication during flight and to allow the ICU to easily command the CWS into a certain mode CWS was realized as a state driven instrument. The different states, the possible state transitions, and the behavior of the software within each state are described below.

Figure 3 illustrates the transitions between the 7 possible states which are discussed in more detail below. Arrows with solid lines denote transitions that require the `setstate` command (either from the ICU or from an EGSE) while arrows with dotted lines represent state changes that are done automatically.

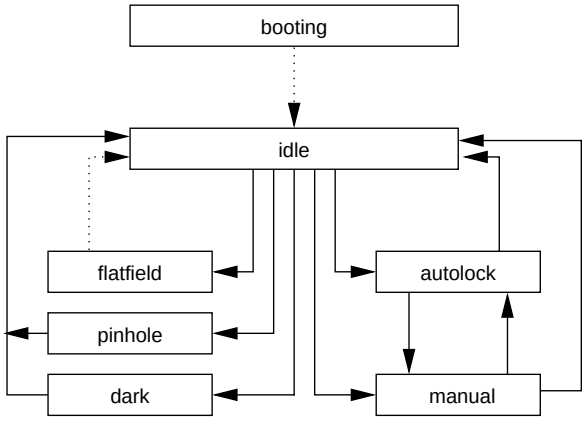

Figure 3. CWS state chart

1. **booting** this state is adopted directly after power-on; it is only left after contact with CW-AO was established; successor state is **idle**; no commands are accepted in this state, hence, no other state except **idle** can be reached from here;

2. **idle** the state reached after a successful power-up and each of the other states; all commands, including state change command are accepted in this state;

3. **flatfield** whenever a `setstate` **flatfield** command is received and accepted CWS changes to this state; it is assumed that the PS does a random walk while CWS remains in this state, but it is solely the responsibility of the ICU or the operator of the appertaining EGSE to ensure this; once CWS completed its flatfielding sequence it changes to **idle** state automatically;

4. **pinhole** state is attained after a `setstate` **pinhole** command; CW-Com stays in this state until it receives a `setstate` **idle** command;

5. **dark** this state is reached after a successful "setstate" **dark** command; as long as no `setstate` **idle** is received CW-Com will remain in this state;

6. **autolock** this is the mode for automated observations; in this mode tip-tilt and focus (M2) loops are running; in case CWS loses its lock it tries to re-lock at the same position; CWS does never leave this state unless it receives a "setstate" **idle** command that requires a state change; note that CWS does not accept any other commands here;

7. **manual** state can be reached either from the **idle** or the **autolock** state; both can also be successor states where the successor state does not necessarily have to be identical to the predecessor state; here every (low-level) command is accepted, where the only check made is the guarantee of the appropriate successor state; the typical usage of this state is to command the CW-AO computer while keeping the loop closed;

4.1 States

This section discusses each of the individual states CWS might adopt. The pre- and postconditions for each state are discussed. Furthermore the behavior within each state is described here in detail.

4.1.1 booting

The state **booting** is adopted once the CWS is powered up. While the CWS is in this state it does not accept any command, it will deliberately ignore each command trying to change the state. While the CWS is in this state, it will power up CoSM, the proximity electronics box, the camera and the CW-AO. It will remain in this state until CW-AO responds to the `status` command, i.e., provides the necessary data that enable the CW-Com to send reasonable HK-data.

The expected time the CWS remains in this state after power-up is 5 minutes. It will automatically change to the idle state thereafter.

4.1.2 idle

This state is intended for manual operation of the CWS. The instrument will not take any action in this state except sending HK data. Whenever the idle state is reached, CW-Com will ramp the tip-tilt mirror and will make sure that the AO loop is open and F2 is set to field stop (light passes to the instruments). The CWS will accept any command including state transition commands in this state.

4.1.3 flatfield

In order to initiate a flatfield calibration, the ICU commands the CWS to the **flatfield** state. The CWS flatfielding procedure consists of the following steps and takes about 6 minutes.

1. ensure AO-loop is open

2. adjustment of filter wheel for optimal light level
   expected time: 180 seconds

3. take the actual flatfield
   expected time: 2 seconds

4. set the F6 position to pinhole
   5 seconds

5. mark and measure the F6 reference spot positions of the Shack-Hartmann sensor that define the perfect wavefront
   expected time 5 seconds

6. set F6 position to field
   expected time: 2 seconds

The CWS does not accept state transition commands while it is in this state and will reach the **idle** state after completion of flatfielding.

4.1.4 pinhole

Taking F2-pinhole images is also commanded by the ICU. CWS moves the tip-tilt mirror to its optical zero position and sets the F2 unit to the pinhole position. The minimum time CWS will remain in this state is 2 seconds. While in this state, CWS does not accept state transition commands except to the **idle** state.

4.1.5 dark

If a `setstate` **dark** command is received this state is adopted. CW-Com commands the F2 unit to the dark position before taking darks. For the CWS the dark procedure consist of a single command and requires about 10 seconds to complete. It does not accept state transition commands while in this state and will only accept a transition command to the **idle** state after completion of the dark procedure.

4.1.6 autolock

This is the state intended for automated observations. A typical observation scenario looks like this:

1. point the telescope to the region of interest
2. take a flatfield (`setstate flatfield`)
3. wait for all instruments to complete their flatfield
4. set CWS to autolock mode (`setstate autolock`)
5. wait for tip-tilt and focus loop to close
6. do actual observation sequence
7. open tip-tilt/focus loop by setting CWS to idle (`setstate idle`)

The behavior of the CWS in the **autolock** state is split into two parts, a startup phase that persists until a lock (tip-tilt plus focus) is achieved for the first time for at least 20 seconds (provided reasonable residuals are achieved) and an operational phase thereafter. During the startup phase a new reference image is taken before each attempt to close the loop. If the CWS loop crashes more than five times in a row or a lock signal from the PS is not received within 150 s during the operational phase (e.g. due to image motion beyond the range of the tip-tilt mirror) it will restart with a startup phase.

Figure 4 shows a sequence of operation diagram for the startup phase. If an arrow is not labelled there is no other choice and hence, this transition is mandatory. The main action in this phase consists of ramping the tip-tilt mirror to ensure a well defined position, changing its position to the electric zero position, and waiting for the "Lock" signal from the PS. We wait for this signal for at most 5 minutes, if it isn't received within this time we try to close the loop anyhow. Closing the loop is done in two steps, first using the tip-tilt correction only and then with focus control in addition. If either of the two fails we start over with ramping of the tip-tilt mirror. A failure condition is defined as not holding a lock for at least 20 seconds or too large residuals ($> 0.02\lambda$ for any of the two axes).

The default scenario during the operational phase should not require any actions besides offloading of accumulated image shift to the telescope, provided no failure occurs, where a failure was defined above. Figure 5 shows the behavior during this phase. In case of a failure we ramp the tip-tilt and wait for a lock signal from the PS. In contrast to the startup phase the timeout is set to 150 seconds here. CWS makes a maximum of 5 attempts to lock using the old reference. If this fails, a "new" startup phase is initiated.

Note that the CWS will never leave the **autolock** state unless it receives a setstate **idle** or **manual** command. Especially if the CWS is not able to get a lock at the position the telescope points to (e.g. at the solar limb) it will remain in the startup phase of the autolock state forever. It is the responsibility of the ICU (or an operator of the EGSE) to monitor that a lock was achieved within a reasonable time.

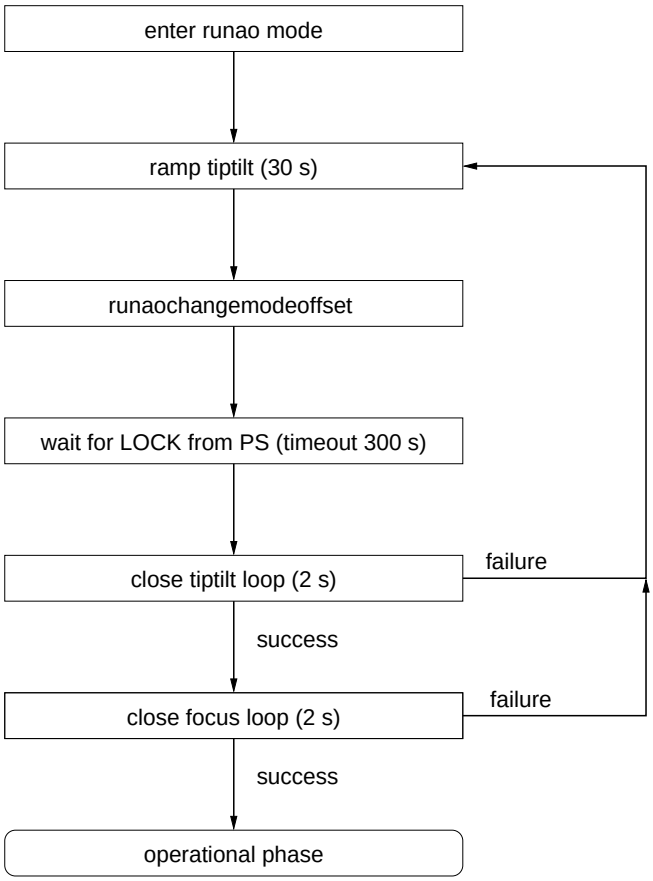

Figure 4. Startup phase of autolock state

5. RESULTS AND OUTLOOK

During its successful flight in 2009, the SUNRISE telescope had by far the highest pointing stability ever achieved on a balloon-borne telescope. During its 5 day flight the flight control software of the CWS consisting of roughly 13.000 lines of C++ code not counting code written for the on-board communication layer operated error free.

The involved partners therefore are in favor of preparing a second flight in 2012 close to the solar maximum, with an additional third scientific instrument that will be defined in the near future. We expect that only minor modifications to the flight control software of the CWS will be required.

ACKNOWLEDGMENTS

The German contribution to SUNRISE is funded by the Bundesministerium für Wirtschaft und Technologie through Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), Grant No. 50 OU 0401, and by the Innovationsfond of the President of the Max Planck Society (MPG). The Spanish contribution has been funded by the Spanish MICINN under projects ESP2006-13030-C06 and AYA2009-14105-C06 (including European FEDER funds). The HAO contribution was partly funded through NASA grant number NNX08AH38G.

REFERENCES

[1] P. Barthol, A. Gandorfer, and M. Schüssler, "The sunrise mission," *submitted to Solar Physics* , 2010.

[2] T. Berkefeld, W. Schmidt, and D. Soltau, "The wave-front correction system for the sunrise balloon borne solar observatory," *submitted to Solar Physics* , 2010.

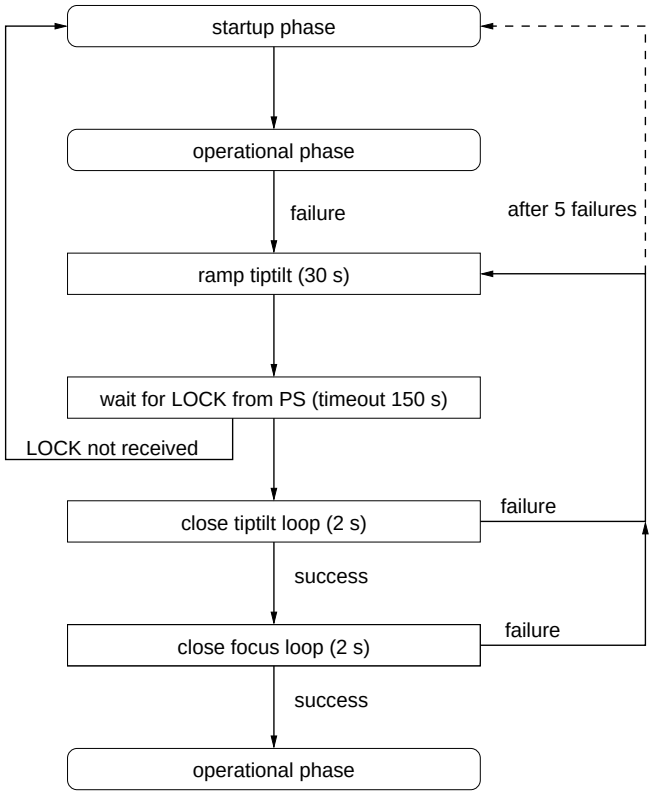

Figure 5. operational phase of autolock state

[3] R. Volkmer, O. von der Lühe, and F. Kneer, "Gregor: the new 1.5m solar telescope on tenerife," in *Innovative Telescopes and Instrumentation for Solar Astrophysics*, S. Keil and S. Avakyan, eds., *Proceedings of the SPIE* **4853**, p. 360, 2003.

[4] Schmidt and Huston, *C++ Network Programming*, vol. 1, Addison Wesley Professional, 2001.

[5] D. C. Schmidt, "Reactor: An object behavioral pattern for concurrent event demultiplexing and dispatching," in *Proceedings of the 1st Annual Conference on the Pattern Languages of Programs*, pp. 1–10, (Monticello, Illinois), August 1994.

[6] T. Riethmöller, "S/w icd between icu and sufi, supos, imax, cw and ps," Tech. Rep. SUN-MPAE-ID-SW000-001, Max Planck Institute for Solar System Research, April 2006.