The PFS Calibration System Controller
II. Software and Operating System

J. Gunn 200215v2 (preliminary)

1. INTRODUCTION

The calibration system will be delivered to ASIAA with interim software installed, to be updated to comply fully with the Subaru requirements at some point.The delivered configuration is fully functional, with all of the observing and safety and keepalive functions implemented, but is not yet fully PFS/Subaru compliant. The code is all written as a set of Bash shellscripts and functions. There are engineering-mode commands as well as a full-up observational system with proper safeguards, and all have necessary features to protect the system components (primarily from overheating) and the observing program from inadvertent operation of the lamps.

The system is built around a standard Raspberry Pi 3B, with a 4-core 1.2GHz ARM v7 processor, 1 GB of memory, 4 USB 2.0 ports, 100Mb/s ethernet. and a micro SD card for file storage with an obligatory small Vfat partition for the boot partition and a single 2GB ext 4 partition for the system and user. It is a 32GB Samsung high endurance card. In addition, it has another such card in a USB reader permanently installed and writeable. Utilities are provided to recreate the system periodically to prevent bitrot on the SD cards.

It has a full installation of Raspbian Stretch 2018-10-09, running Linux kernel 4.14.71-v7+, which is to be considered firmware and never updated. The system is configured to be able to run with an overlay filesystem, so there are no writes to the system microSD card—all disk activity is in memory. Since the Pi does not have a large amount of memory, the writing of large volumes of data is not possible, but the ~800MB of free memory is more than large enough for the calibration application, which never uses more than a tiny fraction of the space.

It is easy to switch between the overlay system and the standard configuration for development, but the system as delivered should not require any development aside perhaps for network configuration and the like.

The function of the machine for the calibration system is quite simple and in normal operation requires only two or three commands, a lamp configuration command which specifies the lamps to be used and the exposure times for them, which turns the lamps and the fans on and performs any necessary preheat (a few seconds for all but the Mercury-Cadmium, which requires 2 minutes for the cadmium spectrum to develop) and waits for a command to begin the exposure timing, a `ready' command whose return status indicates readiness, and that `go' command, which starts the timing and turns off the lamps in sequence and does a general cleanup, and then (optionally) starts a post-exposure cooldown which runs the fans at full power for a period (200 seconds at present).

The engineering commands allow one to run one lamp at a time, and turnthe fans on and off at any selected speed. The lamp brightness is monitored by photodiodes which interface with an ADC on the Pi, and can be logged (which is done automatically in observing mode) or displayed on the terminal. Engineering mode is set by all engineering commands and turns off when requested or automatically 10 minutes after the last invocation to prevent inadvertent disaster.

2. NETWORK SETUP:

The network on the Pi is set to use DHCP, so if it is connected to a network which has a DHCP server, it will get an IP address. It is configured also so that if it is connected to a network without a DHCP server, it will fall back onto a fixed IP address, which for this machine is 192.168.1.200. Thus it is possible to connect the machine to a single other computer or laptop with a crossover cable or a switch, set the IP of the other machine to 192.168.1.something and connect.

If you are already ON a NAT router or any router with a DHCP server, you need some way to find out what IP address you have been given, and you can ssh to the Pi with that IP. The hostname for the Pi is 'pfscalib', which is one way of finding the IP with suitable tools.

Until we get to the observatory, after which the machine will be on a private and completely open network, you will need passwords or (preferably) ssh keys to log in to the Pi. I will provide the password; after you log in, you can add your public rsa key to /home/pfs/.ssh/authorized_keys (but see below on the system--you must have the machine in the mode in which the root filesystem is writeable to do this in a manner which will survive a boot). Since I do not know what the network you will be using is, I do not feel we can relax security quite yet.

Since the calibration Pi is in the prime focus and is connected to the outside world only by ethernet, the health of the network connection is crucial. Once communication is lost, only a trip to the prime focus will allow resetting the machine. We will deal later with problems/hangs caused by code executing on the machine, which we have managed to make very robust by using a watchdog and an overlay file system. The Pi networking, however, is less than perfectly robust, and there is a process starting at boot which watches the network connection and restarts networking if there is a problem detected. This requires a small setup step, which we discuss in the operating system section (5) below.

3. COMMANDS

Log in to the pi as user pfs. The code to run the system is all in /usr/local/bin on the Pi. When .bashrc is sourced at login, the script caliblamps is sourced, which defines a set of control functions for the system. There are, in addition, a set of shellscripts which do various engineering functions, safety functions, etc.

Do an

allstat

Which outputs a status block on the terminal, which looks something like:

```
200107:025556 Fans 0 Time=15
Ne   off v=0.0292
Ar   off v=0.0293
Kr   off v=0.0295
Xe   off v=0.0292
Hg    on v=0.1785
Cd    on v=0.0295
Cont off v=0.0306
```

and contains the following information:

a. UT date and time
b. Current fan speed (0 or 10-99)
c.(sometimes) time into current operational phase
d. Status (on or off) for each lamp
e. measured brightness of each lamp in volts.

The brightnesses are not normalized to any standard value, but normal levels in a dark room with no reflective surfaces near the lamps are typically (if ONLY the one lamp is operating; there is crosstalk among Argon, Krypton and Xenon)

Neon	*0.9*
Argon	*1.0*
Krypton	*0.7*
Xenon	*0.5*
Hg	*1.0*
Cd	*0.4*
Cont	*3.5*

The continuum signal is a sum of continuum fan tachometer output and lamp photodiode output, about 3.1v from the lamp and 0.4v from the fan. See below for details about the fan tachometer.

A. Operating Commands

There are only three of normal interest:

a) lamps lamp1 time1 [lamp2 time 2] [lamp3 time 3] [nofan]

This is the lamp configuration command. It takes lamp names (neon, argon,krypton, xenon, hgcd, cont -- spelling and case are important) and times in integer seconds. The optional `nofan' at the end prevents the cooling sequence and is useful when doing many calibration exposures back-t-back.

*This command configures the lamps and starts them through their warmum sequences. This is just a short wait (2 seconds) for the noble gas lamps, 5 seconds for the continuum lamp, but 2 *MINUTES* for the hgcd lamp. The mercury spectrum takes about a minute to fully develop, the cadmium spectrum the full 2 minutes. The hgcd lamp hardware itself operates in a special high-power warmup mode for 2 minutes upon turnon. If the lamp is already warm, the warmup time is shorter, and the software monitors this by watching the cadmium diode signal and gives the ready signal (see below) earlier. (NOTE: the continuum lamp and the hgcd lamp have conflicting fan requirements and cannot be run simultaneously ---WTH would you want to?-- and this is disallowed in the software)*

At the end of the warmup period, a file /tmp/calibready is created and subsequently the `ready' command returns success. The observing software must monitor /tmp/ for the existence of this file following the issuance of the `lamps' command, or monitor the return status of `ready'. When it is created or `ready' returns success, the system is ready, and the observing sofware can open the spectrograph shutter and, when it is open, issue the command

b) go

when the exposure is started. This command deletes the /tmp/calibready file and starts the timers for the lamps. Each lamp is turned off after the programmed time, and a file /tmp/calibrunning, which is created when `lamps' is called, is deleted. The deletion of this file signals the observing software that the calibration sequence begun when `lamps' is called is finished. Alternatively, there is a command `running' which returns success when running and failure when not. When this is done, the function `coolit' is executed unless the parameter `nocool' has been issued, which runs the fans at full power for a period (currently 2 minutes) in order to temperature-equilibrate the system AND creates a file /tmp/calibcooling which informs the monitor process (see below) that we are still doing something useful.

If it is desired to abort the lamps command at ANY stage, the command

c) stop

Can be issued, which turns off everything and deletes /tmp/calibready and /tmp/calibrunning and /tmp/calibcooling and kills the caliblamps process.

In normal operation, these three (actually, only the first two) are all that is ever necessary (plus `ready' if that is desired for determining state) but there are, in addition, a series of engineering comands. In order to understand how they work, we

must discuss another aspect of the system, namely the calibmonitor process.

At boot, the calibmonitor process is started by /etc/crontab. This process turns off ALL the GPIO ports every 30 seconds unless either the /tmp/calibrunning file exists, which indicates that we are in a `lamps' sequence, or /tmp/calibcooling, which indicates that the postcooling process is executing, OR a file /tmp/calibtesting, which indicates that we are in engineering mode, exist.

*If /tmp/calibrunning exists, the log file /tmp/caliblog is being written to with intervals less than 30 seconds in all phases of operation. The monitor monitors this updating, and if nothing happens in any 30 second interval, it **turns off everything and kills the pfs session**. The cooling routine does not write to the log but touches the log file periodically, so the behavior of the monitor is the same as during the active part of a `lamps' sequence. During engineering mode the monitor does nothing, but the mode itself times out after a period, as we will see below.*

B. Engineering Commands

a) _testing

This command does two things: it creates the file /tmp/calibtesting, whose existence indicates that we are in engineering mode, and starts a process _testwait, which allows the user to work for 10 minutes without the monitor shutting you off; the engineering routines which turn on lamps or fans, namely

b) _neon [on|off]
c) _argon [on|off]
d) _krypton [on|off]
e) _xenon [on|off]
f) _hgcd [on|off]
g) _cont [on|off]
h) _fans [speed|off] speed is % of full, 10-99 or 0

*each *renew* the 10 minute grace period, which exists only to prevent something being left on by an inattentive operator. After this period everything is turned off.*

It is necessary to EXPLICITLY issue the _testing command when engineering work begins--the _neon, etc commands are locked out unless this is done. This is simply a safety issue.

4. NOTES ON THE INDIVIDUAL SUBSYSTEMS

a. The noble gas lamps (neon, argon, krypton, xenon)

These lamps are low-pressure gas lamps akin to neon sign lamps, and, in fact, the power supplies are 4 kV solid-state neon-sign `transformers'. They fire immediately on

power-on and are quite stable after a few seconds of operation. They are custom lamps made by a neon-sign specialist in New Mexico. They should be very long-lived, and we have four lamp units.

b. The mercury-cadmium (hgcd) lamp

This lamp is a special spectral calibration lamp (Hg/Cd10) made by Osram. The bulbs are rare and expensive (several hundred dollars apiece) and the operation is somewhat problematical. After nearly a year of effort spread over about three years, the system being delivered was developed. It is extensively documented in the equipment documentation, but for this purpose it is only necessary to know that it is slow to warm up and the sequence to operate it properly is a combination of hardware and software. When the lamp is energized, it is in a hardware high-current mode to speed the warmup, and this turns off after about 2 minutes. During this time, the mercury output is essentially developed after about a minute, but it takes the full 2 to develop the cadmium spectrum. This is basically because the cadmium vapor pressure is low when the cadmium electrode is cold and it must heat enough for the vapor pressure to be high enough to produce much light in the arc. During the warmup the fans are off, and the fans being on produce enough cooling to prevent the cadmium arc to develop in any reasonable time. After two minutes the high-current mode turns off (hardware) and the the fans turn on (software) at a low level (20 in the lab--on the mountain this number will be higher because of the low air density, but we must determine it by experiment--theory suggests a number like 35, but the heat transfer problem is difficult to solve with any confidence in the result). The cooling fan after warmup is necessary to prevent the lamp overheating and destroying itself. These lamps have very long lives used in the manner in which we will use them, but they must be used carefully. The cooling time is substantially reduced if the lamp is still warm, but it will not reignite if HOT; it must cool for about 5-10 seconds between on periods. During a calibration session in which many calibration sequences are run one after the other, one can use the `nocool' mode and the warmup time is reduced to about a minute, and can be reduced further with some consequences for the lamp lifetime.

c. The continuum lamp

This lamp has a 50W 12V quartz-halogen bulb run in remote voltage-regulated mode. The filament has a color temperature of about 3000K and will last about two thousand hours of on-time. The bulb is in an adjusted assembly, of which there are four, and changing out the assembly is simple but does, of course, require access to the PFI. Changing a bulb IN an assembly is fairly complex, but the procedure is documented. The lamp turns on (half-power point) in 800 ms, and is very near full power in one second.

d. The lamp output monitoring system

The output of each of the lamps is monitored by a photodiode with a filter (for the spectral lamps) which passes a strong line of the element for that lamp.

Unfortunately, the noble gases have line densities which make it difficult to isolate a specific lamp with commercially available filters, so there is considerable cross-contamination, particularly among the heavy gases. We circumvent this by turning on the lamps in sequence; for the filters used here, xenon does not contaminate anything else but is contaminated by krypton and argon, and so is turned on and measured first. Krypton contaminates the xenon channel but not argon, so it is turned on and measured next. Argon contaminates both krypton and xenon but nobody contaminates neon, so it is turned on an measured next, and finally, neon. These are all in one lamp assembly. Unless the measured intensities are roughly correct, a warning is issued.

There are filters separately for mercury and cadmium, though both are produced by the same bulb in the hgcd lamp. This allows monitoring the rather complex operation and behavior of the hgcd lamp, as discussed above.

The continuum lamp's monitoring system is combined with a system which monitors its fan. See the next paragraph.

Please note for ALL the monitor voltages that in the dark with the lamps off, the signals are all of the order of 0.03V. This is simply the saturation voltage of the output transistors of the single-supply `rail-to-rail' opamps used in the electronics. It is not an offset but a floor. A reading of 0.05 is 0.05, not 0.02, but any output less than 0.03 reads 0.03. This may be a different value at the lower temperatures on the mountain. The normal readings for all the lamps are considerably higher than this floor, so it is really of no concern.

d. The fans

The fans cool the three lamp assemblies and the whole calibration system. They are speed-controlled using the hardware pulsewidth modulation system built into the Pi. There are 6 fans, one (including a high-volume double fan on the continuum lamp) on each lamp, two exhaust fans on the calibration system volume and one intake fan into that volume. All fans are wired in parallel (see the system diagram) and so are not controllable individually. Since it is dangerous (to the lamp hardware) to run the continuum lamp without a fan, there is a system which converts the tachometer output of the continuum fan(s) to a DC voltage, which is added to the lamp monitor voltage for this lamp. We check when the continuum lamp is being used by turning on the fan first and seeing if the voltage level is appropriate to the fan speed (normally 50% for this lamp) before turning on the lamp, which results in a much larger signal.

The continuum fan output at speed S is roughly F = .00384*S + .172 volts. topping out at roughly 0.48 at S=80, and falling off a little below S=15 to S=10, below which the fans do not operate reliably and is disallowed in the software.

5. THE PI OPERATING SYSTEM

a. General; the overlay system

The current OS on the Pi (and I propose NEVER to touch it) is Raspbian 9.4, (2018-10-09), running Linux kernel 4.14.71-v7+. It has been modified to run with the root filesystem, in which you write *all* data, as an overlay over the permanent system, which lives on a 32 GB microSD card in a single 5GiB partition. Since the Pi has only 1 GB of memory, this system will crash if you write more than about a half GB of files, but the only file the calibration system writes is a simple text log, and each entry is less than 150 bytes. A calibration session will typically last a few minutes, say 10, and the log is updated every 15 seconds on average, so that is 40 entries, 6 kB; if that is done for each of 25 fields in a PFS night, that is a 150kB log. That should be downloaded and saved each night, but even if not, 3300 nights of data can be stored in /tmp without a problem. If PFS is on the telescope a week a month and we never reboot and never touch the log, we will run out of storage in about 40 years. There is a system watchdog on the pi, and various error conditions such as out-of-memory, too many open files, and several others will cause a reboot. I have been unable to make the system hang despite some effort.

The filesystem /tmp is in memory in any case, and with the overlay the root filesystem is as well, so we never write *anything* to the SD card.

The reason for doing this is severalfold. The micro SD card has a limited capacity for writes, and so will wear out eventually. Worse, the card can be corrupted if there is a power glitch while the system is writing to it, and in normal operation the system is writing a significant fraction of the time that it is doing ANYTHING. With the overlay system, it is impossible to damage the card this way, because it is never written to. Every boot brings the system up pristine.

The downside of this, of course, is that changes you WANT to be permanent are lost. There is a way around this, by configuring the system to boot in normal read/write mode so that you can update the code, deal with network issues, etc, and then return it to protected overlay mode, and there are simple scripts to accomplish this.

This can work because of a small vulnerability in the system. There are two partitions in the Pi system, a small VFAT /boot partition, and the large root partition for which we make an overlay. The system never writes to the boot partition in normal operation, so it is safe even though it is not write-protected. We change the configuration by changing files in /boot which the Pi reads at boot time to create the overlay or not, depending on what we want.

The other storage medium on the Pi, the USB-mounted micro SD card, has a complete copy of the OS and is, in fact, a bootable image in /dev/sda1 (boot,vfat) and /dev/sda2 (rootfs,ext4). There are mount points /mnt/usbboot and /mnt/usbroot for these, but they are NOT automatically mounted. There are also two large partitions on this card, /dev/sda3, mountpoint /scr0 (10GB) and /dev/sda4, mountpoint /scr1 (16GB);

these are also not automatically mounted.

b. The Watchdog

The system is protected by a hardware watchdog timer which is built in to the Pi hardware. There is a daemon running on the system which updates a timer every

fifteen seconds; if the system becomes unresponsive, the daemon will take longer, the timer is not updated, and the system reboots. Given the restricted access to the machine at Subaru, this is a necessary feature. This is tested with a so-called forkbomb, which proliferates processes until the machine bogs down; the watchdog makes it reboot. Writing too much memory fails gently, because there are system limits; writing simply fails, but the machine stays up. Using the supplied scripts never causes this, and a simple reboot will fix it, since it clears the overlay.

c. The Network monitor

There is a process called calibnetwatch which is launched at boot by /etc/crontab which checks on the health of the networking. In order to do this effectively, the system needs an IP which it can ALWAYS reach whatever network it is on. There is a file

/boot/serverip

which MUST BE EDITED to provide this IP address (just one line with the IP address). If that file is missing or if its contents cannot be ping'd, the routine synthesizes an IP address from the Pi's own IP by replacing the last field by .1, which usually but not always works. Far better to provide one. The monitor routine pings that address every two minutes, and if the ping fails, the network daemon is restarted and the interface reactivated. There is a recheck after half an hour which causes a reboot if there has been no network access for that long.

d. Booting into normal (read-write mode)

Since the boot mode is controlled by files in /boot, which is writeable, it is possible to configure those files either for the overlay mode or normal read-write mode. This is done by copying the appropriate files into the two system files /boot/cmdline.txt and /boot/config.txt just before a reboot. The reboot and shutdown files in /sbin have been modified so that the reboot command and the poweroff command both configure the system for overlay mode when it comes up again, so you have to do something special to make it boot into read-write, which is the desired property. The special commands reboot_n and poweroff_n configure it for read-write, but it comes back into overlay when it is booted again unless you boot or powerdown with reboot_n or poweroff_n. PLEASE NOTE that issueing a poweroff command in real operation is deadly, since the machine is at prime focus and we do not have the ability to control its power, and we cannot get it back. There are copious warnings to this effect in the software.

IT IS *VERY* IMPORTANT NEVER TO USE THE READ-WRITE MODE UNLESS YOU NEED TO FOR SYSTEM MAINTENANCE, AND AS SOON AS YOU ARE DONE DO A SIMPLE COMMAND-LINE REBOOT TO RETURN TO OVERLAY.

The command `system' will tell you which mode you are in.

6. THE LAMP SOFTWARE: ORGANIZATION

We have discussed the functionality of the software above, but it is worth discussing a bit about the software organization. Some of this will doubtless change when the code is adapted to the Subaru system. The code is all in /usr/local/bin on the Pi, and all the signalling is done through files in /tmp on the Pi.

Most of the operating code which runs the lamps and does the logging and bookkeeping is in the bash script calibfans, which is SOURCED when user pfs logs in either directly or over ssh. The file contains many lower-level functions as well as the main function `fans()'. This function runs in the mode in which it creates its own subshell, accessed via the environment variable BASHPID, which it determines and stores in the /tmp/calibrunning file. Thus `stop' knows what pid to kill. All the signaling to the external world is via the existence and content of files created in /tmp/, as we have seen. Again, the relevant ones are

a. /tmp/calibrunning	***A lamps command is running; contains the lamps PID.***
b. /tmp/calibready	***A lamps command is running and the lamps are ready***
c. /tmp/calibcooling	***System is in cooldown mode***
d. /tmp/calibtesting	***System is in engineering mode***
e. /tmp/caliblog	***The system log***
f. /tmp/netmonitor	***The network log***

Only calibrunning, caliblog, and netmonitor have relevant content.

The fan control is via a shellscript `fans', which behaves somewhat differently if called from `lamps' or as the engineering script _fans, which is accessed through a symlink.

The environment variables relevant to the system (parameters, filenames, etc) are created and exported in the file `calibhead', which is sourced by `caliblamps'.

There is a script `portsetup' which does all the initialization of the GPIO ports and PWM ports, which is run by /etc/rc.local. All the hardware control is via writing to files in /sys/class/gpio and /sys/class/pwm; this interface is slow but simple and reliable, and certainly vastly more than fast enough for this purpose. As described above, the monitor process calibmonitor is also started at boot, by /etc/crontab; as is the

calibnetwatch process.

The engineering scripts are all separate from the caliblamps script, mostly to avoid touching the latter and introducing bash errors.

7. PHYSICAL MAINTENANCE OF THE MEDIA

Data on SD cards do not last forever. The main and USB cards on the machine are `high-endurance' cards, which means that they will be OK for lots of writes (which we do not do) but not necessarily that the static data on them like the OS will last a long time. It is therefore prudent to rewrite the crucial data on these cards periodically--the cards do not degrade physically over time, but the data can decay, so just as for magnetic tapes, they should be rewritten every year or so. There are several scripts provided to do this:

a. refreshimage

> *This script 'dd's the disk area containing the /boot and / filesystems (and a little extra) to the root of the USB disk, /dev/sda, thus creating a bootable image (if only the Pi could boot from it) on the root of the USB disk. It then. after querying the operator to be sure there has been no trouble, rewrites this image to the root of the SD card. This has to be done while in overlay mode and by the root user; it will not execute if these are not so.*

b. makeimage

> *The third partition of the USB card (the first two are the /boot and / image partitions can be mounted as /scr0. The filesystem is about 25GiB. Backup images can be stored there, and makeimage makes such a backup, named /scr0/pfscalibimage.YYMMDD.img.*

c. getimage imagepathname

> *If you feel that there are errors or have made some mistake and want to regenerate the file system but are still able to boot, this routine will write from an image file created by makeimage to the root of the SD card.*

NB!! these routines all have a hardwired image size of 2000MiB in them, which will accomodate the filesystems /boot and / at creatoon in their partitions /dev/mmcblk0p1 and /dev/mmcblk0p2. If these are resized ever, this will need to be updated.