Upgrading TRINAT Control System

G. E. Claire Preston

Work Term Period: January-April 2015
Program: Honours Physics Co-op, McMaster University
Project: TRINAT (TRIUMF's Neutral Atom Trap)
Supervisor: Dr. John Behr
Location: TRIUMF
Date Report: May 1, 2015

Contents

1 Overview

As a third year Honours Physics Co-op student at McMaster, I was hired for my first co-op work term by Dr. John Behr at TRIUMF to work in the TRINAT group. This work term report summarizes work I have completed up until the end of April 2015, but my total term will be eight months, such that I shall be continuing until the end of August.

My work was focused on TRINAT's neutral atom trap for beta decay. We were operating TRINAT's second atom trap, which is a magneto-optical trap that uses laser light and a magnetic field to confine atoms to a region only a couple of cubic millimeters in volume. Laser cooling slows the atoms and lowers their temperature in a phenomenon called optical molasses. With the frequency of the laser light detuned slightly below the atomic resonance, any motion of an atom towards a laser source increases the frequency of the light towards resonance due to Doppler shift, causing more photons to be absorbed by the atom such that it absorbs their momenta and slows along that axis. With the magnetic quadrupole field being zero at the center of the trap, for small displacements from this region, the magnetic field strength increases linearly. This causes a Zeeman shift in the atomic energy levels, shifting the levels closer to the laser light frequency, such that more photons will be absorbed and the atom will be kicked back to the center due to the impulse they impart upon it through absorption.

This apparatus is used to trap unstable atoms, most recently potassium 37, which then undergo beta decay producing a recoil daughter nucleus, a beta particle, and a neutrino. The atoms are highly polarized such that their spins point in the same direction along the z axis due to optical pumping performed with circularly polarized light. The directions in which the decay particles travel are predictable due to the high degree of polarization, and the majority of the escaping products are captured using large electric fields. The momenta of all particles can be found, that of the nucleus and beta particle directly from measurements, and that of the neutrino indirectly by using the conservation of momentum law and subtracting off the momenta of the other two particles and comparing with the original total momentum.

The main particle of interest in this decay is the neutrino. The Standard Model indicates that only left-handed neutrinos can result from beta decay. TRINAT is one of the projects searching for the possible emission of right-handed neutrinos that would provide evidence for new physics beyond the Standard Model. Initially, a potassium 37 nucleus has a spin projection along the z axis of $\frac{3}{2}$, a value that must be conserved. After the decay, a positron is emitted downwards, contributing a spin projection of $-\frac{1}{2}$. According to the Standard Model, a left-handed neutrino would be emitted, and this would go upwards if this were the case, contributing a spin also of $-\frac{1}{2}$; however, this would mean that for the spin projection along the z axis to be conserved, the spin of the argon 37 recoil nucleus would have to be $\frac{5}{2}$. In reality its ground state spin is $\frac{3}{2}$, so this decay is actually forbidden. If these decays are observed, it becomes evident that a right-handed neutrino, with a spin projection along the axis of $+\frac{1}{2}$ must have been emitted such that the spin of the argon 37 would be correctly $\frac{3}{2}$. In truth, this happens about 1% of the time, but if statistically significantly more events were detected, this would point to a violation of the Standard Model statement. TRINAT is only one of the experiments looking for these types of decays.

The atom trap is currently in the upgrade stage. In part this is to increase the accuracy and precision of measurements to be competitive with other experiments measuring similar quantities. My main focus during this four-month term was on aiding in the upgrade of the trap's real-time control system. The trap and old Electrim cameras are currently controlled by a DOS program on a severely aging Windows computer, for which there is no backup hardware if it fails. This term we have been working on configuring a new camera for trap imaging and replacing the old computer with a Raspberry Pi computer running a real-time Linux operating system.

In this report I will outline my major tasks completed or still in progress this term. First

I will discuss our attempted configuration of a Point Grey Flea3 GigE camera to replace the old Electrim cameras, and then our change to a Point Grey Flea3 USB 3.0 camera. I will discuss the setup and measurement of exposure time of the camera, as well as my advances in programming it to perform functions useful to us using Point Grey's FlyCapture Software Development Kit, and externally triggering and strobing it. Next I will discuss our usage of the new camera to perform an absolute calibration of the recoil nucleus MCP position data, as well as a measurement of the camera scale with distance given the particular lens setup. After that I will discuss configuring a Raspberry Pi computer to improve real-time control capability, in hopes to replace the DOS control system. This has involved testing latencies of the B+ and 2 model with a Linux kernel that has a real-time (PREEMPT_RT) patch applied. These tests were performed using the GPIO pins and an ABElectronics DAC-ADC connected to an oscilloscope in various setups, most successfully in using the bcm2835 library. I will also mention the usage of the Pi to control a LabJack U3-HV board.

In summary, the main results are presented as follows:

- Flea3 USB 3.0 camera can be externally triggered at 2.25ms intervals with a minimum true total exposure time of $80\mu s$, can strobe to indicate when it triggers, and can have a region of interest specified for capture
- Knowledge of trap absolute position has improved by a few millimeters, having been determined more accurately using images from new camera in comparison to projection of MCP data, with potential to obtain accuracy of position within approximately 0.1mm once it is redone with the system better focused
- Raspberry Pi 2 running a 3.18 real-time Raspbian Linux kernel has been observed to have a maximum latency time of $10\mu s$ per GPIO/SPI instruction using the bcm2835 library and real-time scheduling, can also control a LabJack U3-HV, and is therefore being considered a viable option for controlling the trap

2 Camera Setup

With a desire to upgrade the system came a desire to set up a better camera to image the trap, replacing two old Electrim cameras. The first goal was to find a camera with an exposure time of less than $10\mu s$, to be able to catch the atoms during a flipping of polarization and image them in two different polarization states. In the end, the camera we currently have does not achieve this, though it brings our minimum possible exposure time down to $80\mu s$. The Firefly MV camera that is a camera installed after the Electrim cameras claims a minimum exposure time of $31\mu s$, but this has not been tested, and the current firmware only allows a minimum time of $62\mu s$ to be set.

For a while we investigated the difference between CMOS and CCD sensors. In the end we didn't reach a clear conclusion as to which is better as CMOS has undergone so much development in recent years that apparently certain sensors have quality on par with CCD. Generally, it seems that in earlier years, CCD exhibited higher quality, lower noise images, and higher light sensitivity due to more of the chip being dedicated to light intake instead of transistors, but it is unknown if this is still the case in most modern CMOS detectors [1].

2.1 Point Grey Flea3 GigE FL3-GE-03S1M-C

Our first camera we tried to make function connected via ethernet, had a CCD sensor and a claimed $30\mu s$ minimum exposure time. For a few weeks we attempted to get this camera to work with the FlyCap program and the behrlt5 and trinatlt2 computer (Info file at [2]). The IP address was registered with TRIUMF, and we were able to detect the camera using the FlyCap program, but could not get it to stream images, getting errors indicating possibly dropped frames. We tried using a purchased ethernet switch between the laptop and the camera and fast ethernet cables, but still were unable to make the camera operational with the demo program, even with adjusting packet size and packet delay. My main suspicion is that perhaps the ethernet card on the computer we were plugging the camera into was not fast enough to handle the data flow. This is mainly because Point Grey strongly recommends a PCIe card to use with it that plugs into a computer tower [3], but we were using a laptop.

2.2 Point Grey Flea3 USB 3.0 FL3-U3-13Y3M-C

This camera was purchased around the end of January in hopes that it could be made to work easier than the ethernet camera, connecting via USB. It has specs including: sensor size: 1/2", pixel size: $4.8\mu m$, minimum exposure (claimed): $6\mu s$, sensor type: CMOS, quantum efficiency ($770nm$): 36% [4] [5].

It exhibits some strange qualities with Linux, in that sometimes it thinks it is a USB 2.0, and sometimes a USB 3.0 camera. To get it to read properly as a USB 3.0 camera, the following sequence was found to work: Unplug camera, restart both camera and computer, then plug in camera. On the Point Grey website, we have been warned to not update the firmware while the camera thinks it is a USB 2.0 camera [6].

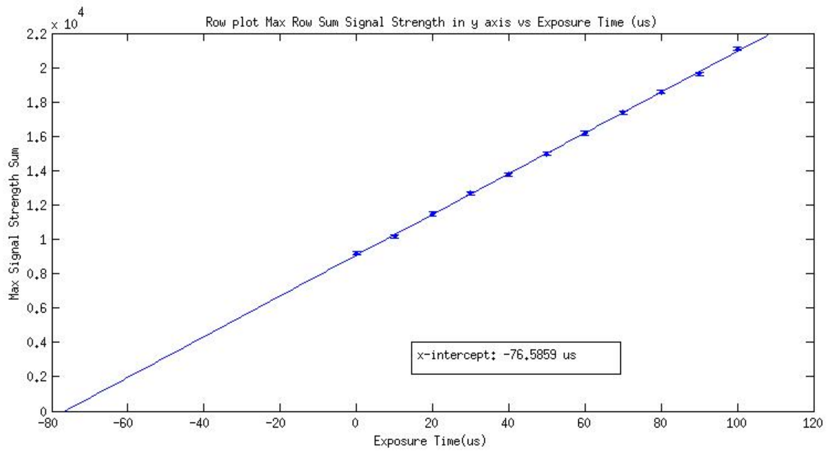
2.3 Exposure Time

The claimed exposure time of this camera was a minimum 6 microseconds. It originally was running firmware version 2.07.3, but this was upgraded to experimental emailed firmware from Point Grey version 1.45.2 to decrease the minimum exposure time able to be set. If desired the old firmware can be re-installed, it is backed up as listed in the files list. Originally the demo FlyCap program could only be set to a minimum of $10\mu s$ exposure, but this was decreased to a minimum of $0\mu s$ with the installation of the new firmware. Two tests to measure true exposure time were performed.

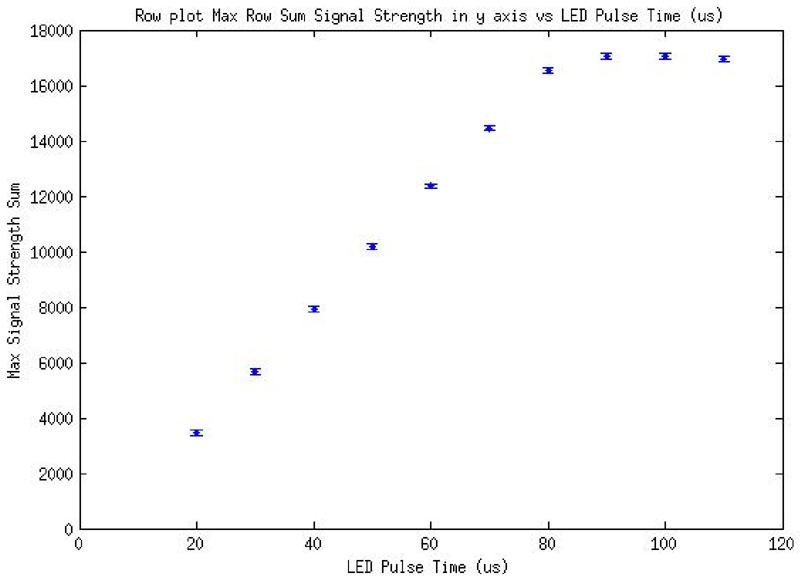
When we observed that exposure times seemed to be longer than those claimed by the Point Grey site and the FlyCap program ($95\mu s$ vs displayed $10\mu s$, with supposed minimum exposure time of $6\mu s$ [4]), we contacted them. We were introduced to a concept coined the 'frame-overhead-time' by Point Grey Support, characterizing a time where the sensor was supposedly done capturing the image, but still exposing to light, apparently a problem they had also measured with this sensor. They claimed they had measured $93\mu s$ for our situation and $40\mu s$ when the frame rate was above 148fps.

In the first test, an LED was hooked up to a Phidget board and run on berhlt5. It was programmed to flash on for periods of 10-120μs in intervals of $10\mu s$, and the camera was set to a minimum exposure of $10\mu s$ with the old firmware. Gain was set to 0. Images were taken using ASyncTriggerEx, so in external trigger mode. A projection of the sum of the pixel values along the horizontal was taken, and the peak values were obtained, using MATLAB and the tif images. Peak intensity values were plotted against LED pulse times, and observed to flatten out around $95\mu s$, setting this as the minimum exposure time, in comparison to Point Grey's claimed result of $93\mu s$. After the new firmware was installed, this was reduced only to around $85\mu s$, now that the minimum time could be set to $0\mu s$ instead of $10\mu s$ as in Figure 2.1b so it was concluded that the minimum exposure time, or 'frame-overhead-time' was actually $80\mu s$. The only effect observed from the firmware was to reduce the exposure times by $10\mu s$ since the minimum setting became 0 instead of 10.

In the second test, the LED was kept on constantly and the exposure time was varied. These results were obtained using CustomImageEx, in continuous capture mode, and saved as tif files. It was noted that it took 2-3 images for the different set exposure time to take effect (this was noted on the Point Grey site [7]) such that only the 3rd or 4th image was taken. Again using MATLAB, the sum projection along the horizontal was taken and the maximum intensity values were plotted against the exposure time. The x-intercept was taken as the true frame overhead

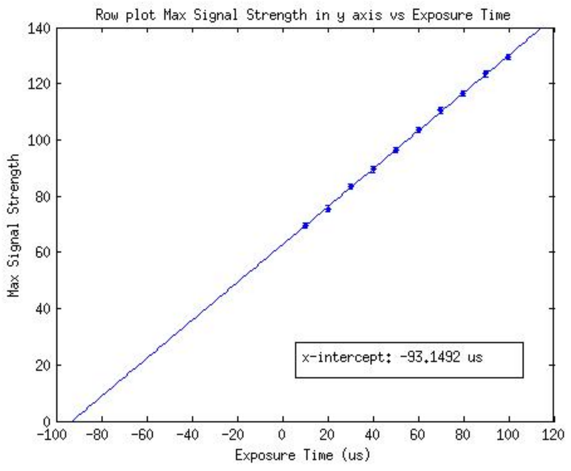


(a) New firmware FL3-U3-13YM3 Testing Exposure Time (x-intercept) for 1280x1024 pix in continuous capture mode by varying camera exposure time

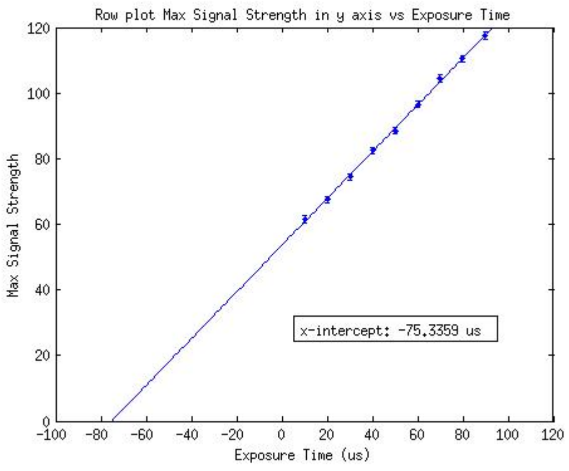


(b) New firmware FL3-U3-13YM3 Testing Exposure Time (x-intercept) for 1280x1024 pix in external trigger mode by varying LED pulse length

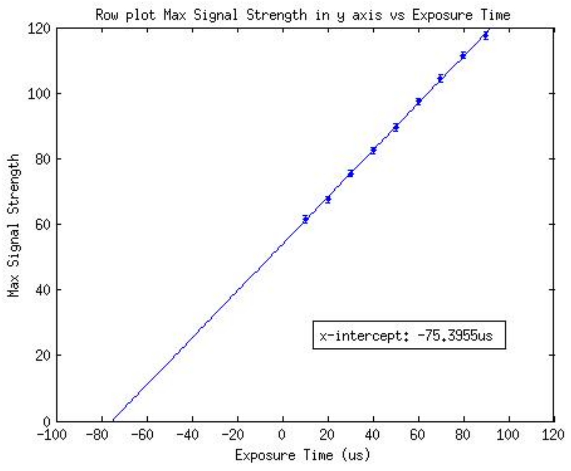
Figure 2.1: Measuring total exposure time with new firmware



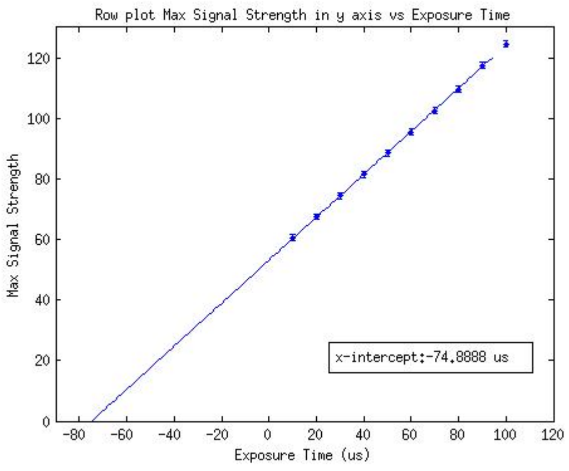
(a) Region Of Interest: 1280x1024 pix



(b) Region Of Interest: 560x400 pix



(c) Region Of Interest: 400x400 pix



(d) Region Of Interest: 300x300 pix

Figure 2.2: Old firmware FL3-U3-13YM3 Testing Exposure Time (x-intercept) for different regions of interest in continuous capture mode by varying exposure time

time and found to be approximately $76\mu s$ for the old and new firmware, slightly less than the other test (Figure 2.1a). This is attributed to possible difference between frame overhead time in the two capture modes: triggering and continuous capture.

Point Grey support claimed in their email that due to a sensor clocking mechanism, when the camera was run at rates higher than 148 fps, they obtained a reduced frame-overhead-time of $40\mu s$. The only way I could get the camera to increase frame rate was by reducing the region of interest. This increased the set frame rate to above 148 fps, though the actual frame rate was not known. It could have been found with some clever programming, but it was a bit beyond me. Even running in this mode, we still observed a frame overhead time of around $80\mu s$. In continuous capture mode, with the old firmware, and exposure time set to $10\mu s$, minimum exposure time was seen to decrease as region of capture decreased, perhaps due to faster turnaround time in reading out fewer pixels (Figure 2.2). In external trigger mode, it was determined that the frame rate did not seem to apply and therefore no method could be devised to achieve a frame rate over 148fps rate in this mode.

2.4 Programming

Programming of the camera was performed using the FlyCapture Software Development Kit, developed and maintained by Point Grey. This offers a closed-source FlyCapture library full of functions to be used with their cameras. I learned to use functions including setting gain, exposure time, capture speed, capture mode (continuous capture vs asynchronous trigger), and activating strobe, in C/C++ routines. These functionalities are fairly well documented in my C++ programs located on trinatlt2 as listed in the File List. Some general program notes:

- One strange property of this camera is that it is observed that up to 3 pictures must be taken for a different exposure time to take effect [7]
- More information about selecting a region of interest can be found at [8]
- The pressing of the ’q’ key to exit a picture-taking loop has also been added, as noted in the file list
- cp3AsyncTriggerEx is my most updated version of externally triggered capture routine. It uses the dynamic memory allocation process I wrote to avoid delays from storing images in the hard drive and keeping them in memory
- cp4CustomImageEx is my most updated version of continuous capture mode routine, and it immediately converts and stores images on the hard drive as image files, such as tif

2.4.1 Dynamic Memory Allocation

I rewrote John Behr’s original code for storing pictures in memory as they were taken, using explicit array declaration and allocation on behrlt4 for use with the Firefly MV camera, into code using dynamic memory allocation classes and functions such that the number of pictures did not have be set at the start of execution. This was tested for memory leaks, including with a downloaded program called valgrind, which shows the user the amount of memory allocated and freed during the execution of a program. The header file contains a datalist class, a data item class, functions, constructors, and destructors to correctly manage the memory allocation. This functionality has been implemented in all most up to date versions of my programs as listed below, and a version has also been added on behrlt4.

2.5 Triggering and Strobing

This camera comes with both software and external hardware triggering ability. It could also easily be configured to send out a strobe pulse upon completing the capture of a photo. This can be done in FlyCap or with a C/C++ program - my most updated version is cp3AsyncTriggerEx.

If a pulse is sent to the camera (strength according to GPIO logic specifications, 2 V seemed to work fine, less than 3.3 V), and it is correctly set in high polarity trigger mode, it will only take a picture on this trigger. More info is at [9] and the technical reference manual [10]. The trigger pulse is sent to GPIO 0 (black), strobe outputs from GPIO 1 (white) and we were using the opto-isolated ground for our grounds (blue). We had to add a pull-up resistor in order for the output to work and remain within current limits for the output circuit according to Figure 2.3.

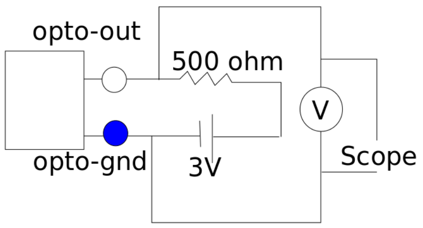


Figure 2.3: Pull-up resistor for Flea 3 strobing

The maximum frequency of external triggering capability was found to be dependent on the size of the region of interest. At 1280x1024 pix, full size, it could only take an image every 12.5 ms. At 304x300 pix the minimum interval for external triggering for which the camera could trigger on every pulse was 2.25 ms, and at 208x200 this was reduced to 1.5 ms. It was considered that triggering at faster rates might force the camera to have a shorter frame overhead time, but the camera would not trigger on more frequent signals, only triggering on every other pulse. As indicated by Point Grey, the camera may not be able to be triggered at maximum frame rate. Trigger Mode 15 was also tried, where the camera is triggered once then goes into something like continuous capture mode, but this proved to have no better results.

2.6 File List

Folder 1: Flea 3 USB 3.0 Continuous Capture Mode Code

trinat@trinatlt2:/home/trinat/flycapture/flycapcode/src/cp2CustomImageEx/
cp4CustomImageEx.cpp
Last updated February 24, 2015

- based on CustomImageEx.cpp example located in ../CustomImageEx
- writes first k_numImages images into tif files in continuous capture mode, as it must take 2 images for set exposure time to take effect, and after that goes through dynamic memory allocation loop storing values as arrays in memory then writing as numbers to file
- includes dynamic memory allocation capability
- includes ability to use key press to exit picture-taking loop
- can change region of interest by adjusting k_width and k_height, and it automatically centers image in ROI if no offset is indicated. Can change offset by changing k_xOffset and k_yOffset. ROI x dimension must be multiple of 4, possibly also y dimension
- can set number images to capture, delay between each image, exposure time, frame rate, gain (set to 0), can enable strobe, timestamping (uses first few pixels)
- Due to key press functionality, if the program is halted in the middle of the dynamic memory allocation loop, the terminal will still not prompt/ not show typed text, so just press 'q' to exit

fire_labjack_2cp_methods.cpp and **fire_labjack_2cp.h** contain the classes and functions for dynamic memory allocation of images.

Folder 2: Flea 3 USB 3.0 External Triggering Code

trinat@trinatlt2:/home/trinat/flycapture/flycapcode/src/cpAsyncTriggerEx/
Note the separate makefiles for each program
cp3AsyncTriggerEx.cpp
Last edited February 17, 2015

- includes dynamic memory allocation changes as well as the ability to use key press to exit loop
- can use software or hardware trigger depending on whether #define SOFTWARE_TRIGGER_CAMERA is commented out
- very similar other features to cp4CustomImageEx including that it takes k_numImages tif images before looping starts
- strobe enabled

cp2AsyncTriggerEx.cpp
Last edited February 17, 2015

- does not include dynamic memory allocation, just stores pics as TIF's

Folder 3: Files for Firefly MV camera on behrlt4

behr@behrlt4:/home/behr/flycapture/src/CustomImageEx_cp/ **fire_labjack_3cp.cpp, fire_labjack_2cp_me**
Makefile_fire_labjack_3cp, and fire_labjack_2cp.h
Last updated

- These are the new files implementing new dynamic memory allocation for use with Firefly MV camera
- similar to cp3CustomImageEx, doesn't implement ROI, can set exposure time, framerate, gain
- still contains functions to interface with LabJack board
- contains ability to quit with key press of 'q'

fire_labjack_1cp.cpp is the original program without the dynamic memory allocation implementation

Firmware Backups for Flea 3 USB 3.0

Backup of old firmware: trinat@trinatlt2:/home/trinat/flycapture/**fl3-u3-2.07.3-00.ez2**
Backup of new firmware: trinat@trinatlt2:/home/trinat/flycapture/TestFirmwareShortExpTime/**fl3-u3_1.45.2.ez2**

2.7 Links

1. http://www.ptgrey.com/KB/10201
2. http://www.ptgrey.com/flea3-03-mp-mono-firewire-1394b-sony-icx618-camera
3. https://www.ptgrey.com/tan/10360
4. http://www.ptgrey.com/flea3-13-mp-mono-usb3-vision-vita-1300-camera
5. http://www.ptgrey.com/support/downloads/10110
6. https://www.ptgrey.com/KB/10127
7. http://www.ptgrey.com/KB/10086
8. http://www.ptgrey.com/KB/10085
9. http://www.ptgrey.com/tan/10591
10. http://www.ptgrey.com/support/downloads/10120

3 Absolute Calibration Trap Position

One task attempted using the new camera was to perform an absolute calibration of the rMCP (recoil nucleus multi-channel plate) data relative to the camera. Seeing as the distance traveled from the trap to the plate is important to be able to find the momentum of the recoil nucleus, the absolute position of the trap is important.

Lenses on and in front of the camera were set up as follows: a 40 mm lens screwed onto the camera, followed by a lens with a focal length facing the camera of 630mm and facing the trap of 627.1mm. An infrared filter was placed in front, closer to the chamber. The distance between the close edge of the 627.1mm focal length lens to the center of the chamber, where the trap was thought to be, was 630mm. This should have brought the trap light to a focus at the camera sensor, but this was not the case as was found in the absolute scale calibration. It was suspected that the 40mm lens was too close to the camera sensor.

Runs 767, 768, 769, 771, were run with the horizontal field current at 0A, +10A, 0A, -10A, and the trap was observed in camera images taken during this time to move approximately 3 mm towards and away from the first trap with the positive and negative currents respectively (left and right as seen through the viewport). Runs 772, 773, 774 were performed with the Z coils at (top/bottom) +2.0A/-2.1A, -2.1A/+2.0A, -4A/+4A, and the trap was observed to move vertically upwards approximately 1.5 mm in the first run, and downwards 1.5 mm and then 3 mm according to the camera images. These images in particular exhibited 2 observed states, one where the trap was confirmed to still be in its ”0” position, and images taken of this state were dim. The second state was bright and exhibited a shift related to the current difference and direction. One image of run 774 was taken with both these states observed. These shifts were assumed to be due to the atoms being transferred by the push-beam during the dim stage.

3.1 Data Analysis

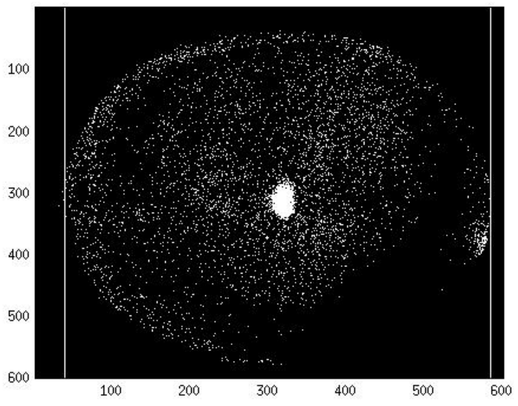
I extracted, manipulated, and analyzed data using MATLAB on trcomp. Projections of the MCP and camera image data were centered and then the trap positions were compared to determine how well aligned the mcp data was with that of the trap.

Presented are plots with Melissa's new calibration done March 17. The plots are for horizontally and vertically shifting the trap as described, with the 630mm distance from the f=627.1mm lens face. Also included is a plot of scale against distance obtained from imaging a grid scale at different distances. Scale used to convert camera pixels was as obtained from calibration with grid at distance of 630 mm, not from averaging hoop sizes, as the former was deemed more accurate. Scale obtained was 0.0801±0.0006 mm/pix, as outlined in next section.

Given the consistency of the shifts and the well-fitting of the plots, I believe that if the camera were better stabilized to prevent moving and the relative trap positions were determined, this could serve very well as an absolute calibration of the trap position to within 1 pixel at the correct focus distance, once the lenses and distances are corrected, but I am unsure what the scale would be with new lens arrangement.

Ion MCP data used was from fancymap_ind plots on trinatdaq as listed in file list. This data has been approximately normalized according to values of time and frequency estimated from Time/Hz graph in ROOT to approximate intensity and amplitude. It has further individually been scaled and shifted vertically, to better visually compare the plots, as requested.

Center for MCP data was taken as the average of the edges of where events started to appear, supposedly at the inner limit of the ceramic ring. Edges were defined as where the value was 0.1% of the max value for horizontal and vertical top, and 0.01% of max value for bottom in vertical (due to plate's low efficiency at bottom right), as pictured in Figure 3.1. Uncertainty due to this and centering is estimated at 0.1 mm (≈1 pixel), for image 600x600pix, 89.85x89.85mm.



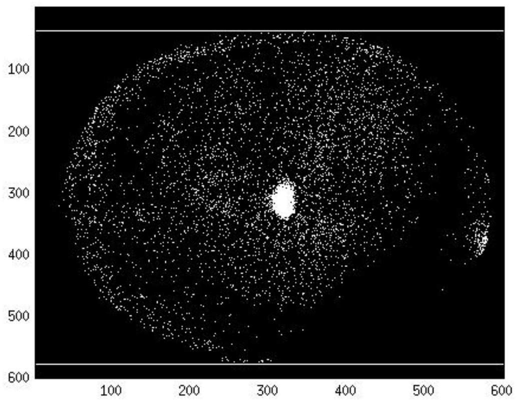


Figure 3.1: Bounds for edge selection run 769 horizontal and vertical MCP data

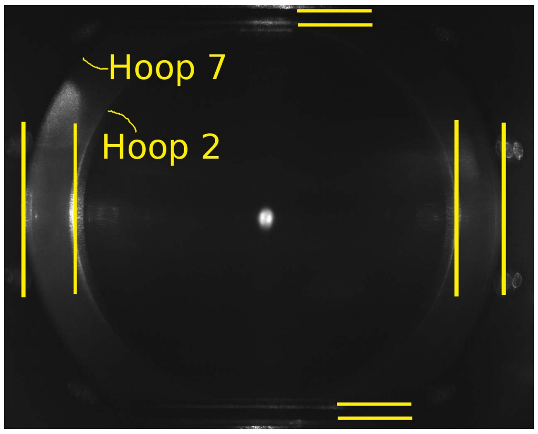


Figure 3.2: Camera image run 769 showing edges selected for centering in horizontal and vertical for convenience and

The trap center from camera images was obtained as average position of hoop edges. Error can be incurred from this centering in selection of hoop edges. In vertical direction, as hoop edges were not visible due to magnification, edges of electrostatic plates in between hoop2 and hoop7 distances were used for vertical centering, possibly increasing error. Uncertainty of center due to estimation of edges and centering is estimated at 1 pixel, possibly more for vertical. See Figure 3.2. Due to 0.8% estimated uncertainty on scale, this translates to estimated 0.3mm uncertainty. This can be improved through better precision in measuring the scale once the camera is focused correctly.

From the camera images and photoion mcp data, I have manually selected the portion of the image that encloses the trap and used this data in the plots, projecting only the strip of area in the vertical or horizontal where trap is present.

Note that run 767 was not included in the analysis and run 769 was used as the '0' run.

Around mid-April, when Melissa was attempting to align the MCP data with the camera data, she discovered a mistake I had made in my vertical plots, in that I had not noticed that my MATLAB code had read the vertical values in reverse. This essentially shifted both the MCP data and my camera projection values by about 1mm. This has been corrected for in this analysis.

3.1.1 Horizontal Shifts

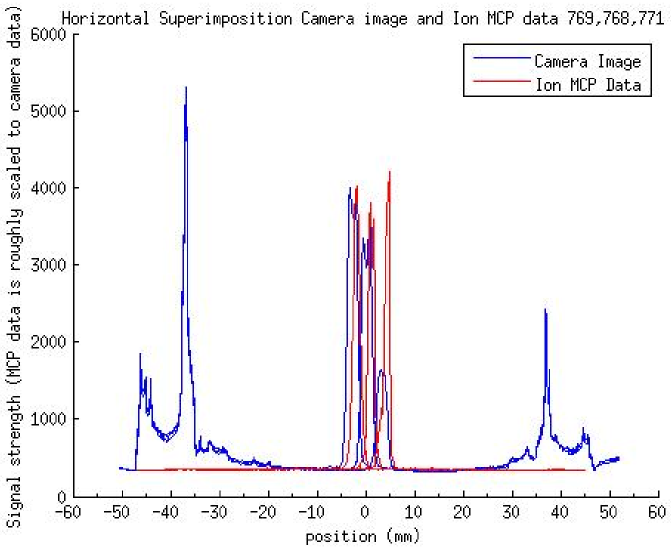
Displayed are original plots comparing projections of camera and MCP data, after vertical scaling adjustments. First see result after centering of camera image and MCP data according to edge selection. Second, see MCP centered data shifted by 10pix=0.801mm to the left. This corresponds to a shift of that amount to the right of the actual MCP data, because the centering reverses the direction of the shift. Therefore it is assumed that to make the two projections match, the MCP data must be shifted around 0.8 mm to the right. The difference between the distances on either side of the trap between the edges of hoop7 and hoop2 in the image was about 7.67pixels 0.61 mm, indicating slightly imperfect centering, which could at least partially accommodate for this. This is predicted to have a maximum influence of around 0.3 mm on shifting the camera projection, which is around the same value as the uncertainty due to scaling and centering. One note the third peak in the camera data was a rather dim image, so the peak has a small amplitude.

3.1.2 Vertical Shifts

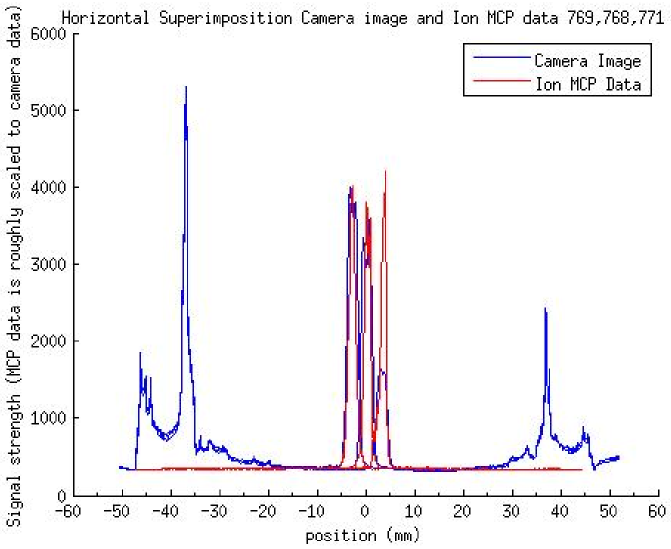
Displayed are plots centered and after scaling. The two blue datasets in runs after 769 correspond to the 'bright' (after transfer) and 'dim' (during transfer) images that were observed. As can be seen, the ion mcp data seems to reflect a superimposition of these two states, where the large peak corresponds to the bright stage and small peak to the transfer stage. The position of the transfer images is verified to be at the position of the 0A 769 run as pictured after runs plots. In title, first number corresponds to current in top coil. The data appeared well aligned without adjustment between the camera and MCP data.

Run 769 (0A/0A), Run 772 (+2A/-2A), Run 773 (-2A/+2A), and Run 774(-4A/+4A) only camera image with 'both' dim and bright stages - are compared with MCP projections in Figures 3.4a-3.4e, with mcp data vertically scaled and shifted to compare shapes. For Figure 3.4d, there are three blue plots shown, corresponding to the dim and bright images, and the medium height plot corresponds to the image where both these stages were able to be seen. This medium height plot is given again in Figure 3.4e alone. This I see as good evidence of the bright/transfer superimposed states being expressed in the ion mcp data. In Figure 3.4f run 769 (tall peak) and the 'dim' stages are given for each run, and can be seen to be in the same approximate vertical position, supporting the idea that the 'dim' stages are during transfer.

Figure 3.3: Horizontal comparison between MCP and camera data, left: 768 (+10A), center: 769 (0A), right: 771 (-10A)

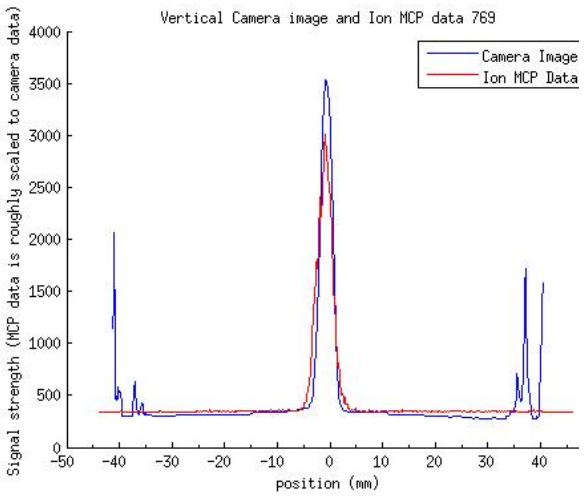


(a) Horizontal centered data, unshifted

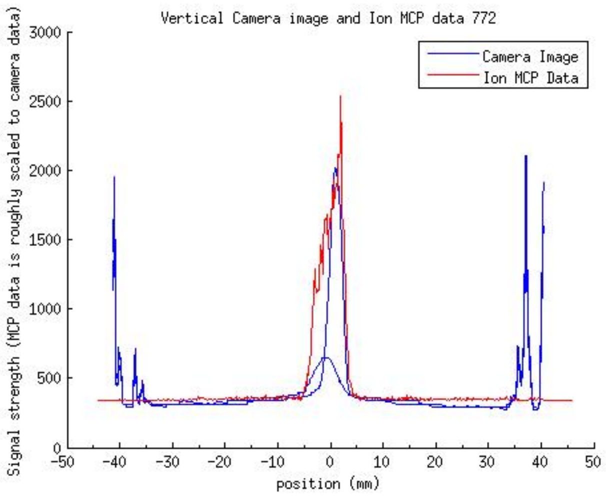


(b) Horizontal centered data, MCP data shifted 10 pix=0.801mm left, corresponding to shift in MCP data of 0.801mm right

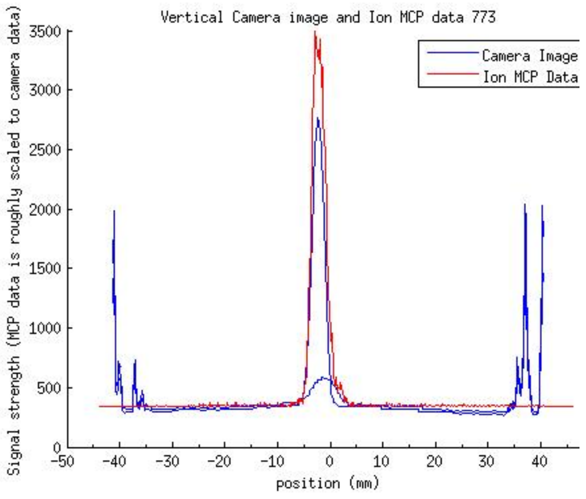
Figure 3.4: Vertical data comparison between vertical MCP and camera data



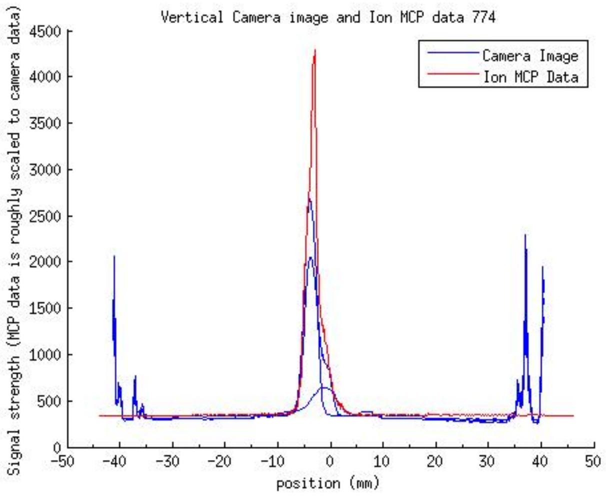
(a)



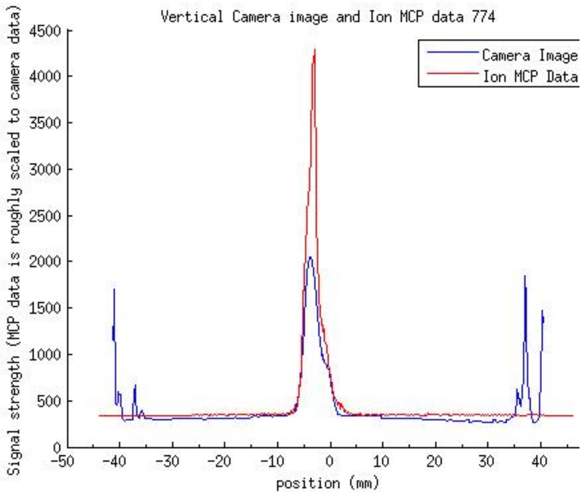
(b)



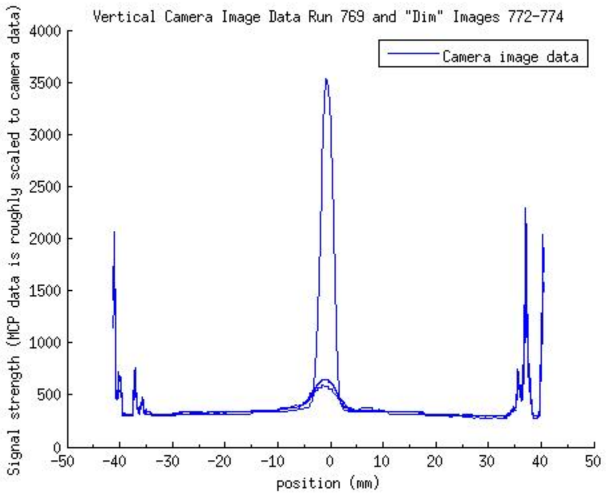
(c)



(d)



(e)



(f)

3.2 Camera Scale and Focus Calibration

We wanted to check that the camera was indeed focused at the correct point and that the scale scaled monotonically with distance from the camera. With the same lenses and IR filter, this measurement was performed using the camera, a 90-degree mirror, and optical rail mounts, without moving the setup from its mounted position in front of the vacuum chamber viewport. A calibration grid with squares of 4mm width measured accurate to approximately 5% was imaged at several different distances from the lens setup. The camera was found to be actually focused at 701±5mm, while measurements were taken at 630mm. This will be corrected before more calibration is attempted.

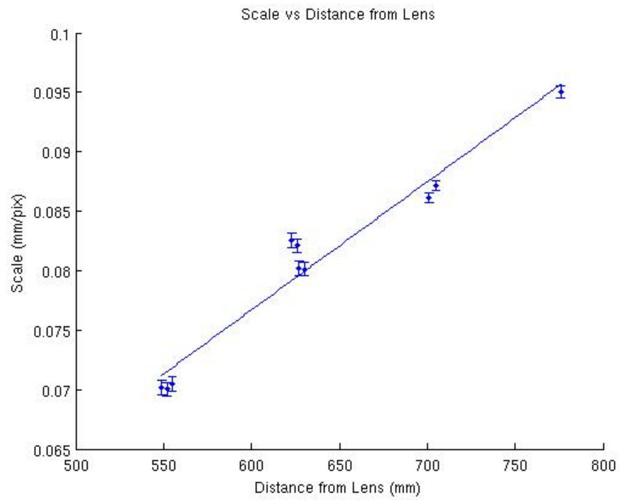


Figure 3.5: Measurement of camera scale vs distance from 630 mm lens

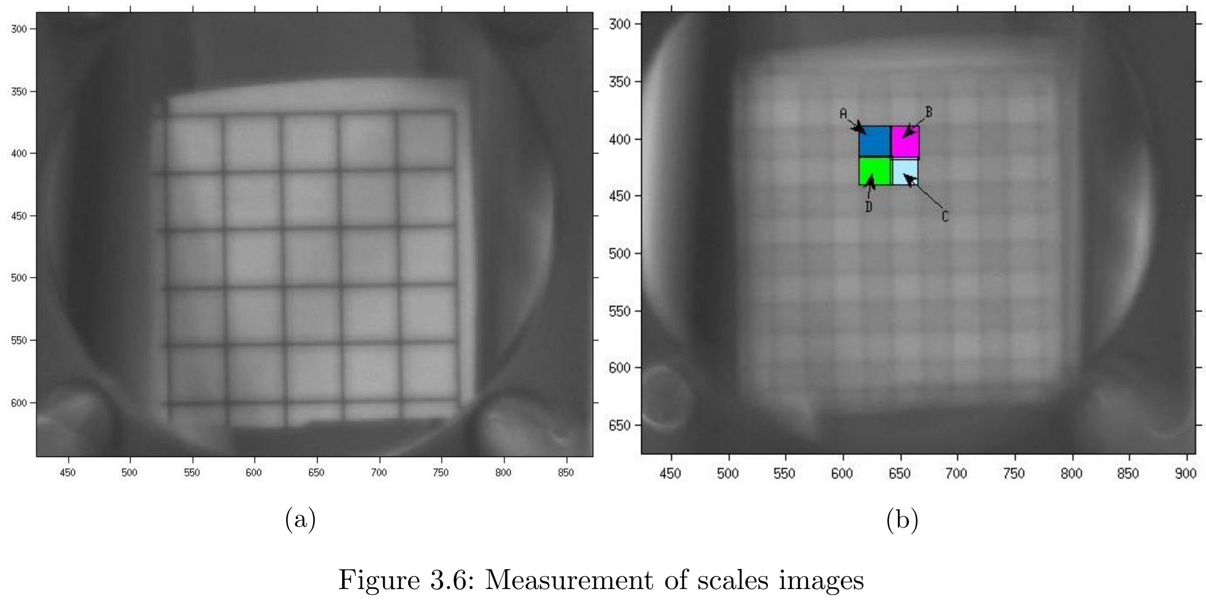


Figure 3.6: Measurement of scales images

The scale was found to be relatively monotonic with the distance for the camera in Figure 3.5. These distances for measurement were chosen because they corresponded to locations of Hoop 2, Hoop 7, and the trap, with respect to the camera's supposed and actual focus. Scale measurements were made using MATLAB and its ability to display the coordinates of a selected pixel. Scales were obtained for the less blurry images by measuring the height and width of 8 squares in pixels as in Figure 3.6a, then averaging them to get the scale in mm/pix. For these, the x and y scales, before averaging, were observed to be quite similar to within 0.0001mm/pix. For the blurrier images, refraction edges of each box could be seen. Four boxes were selected, as in Figure 3.6b, and side lengths were measured to the best of my ability. The width of the box was taken as the average of the sum of widths of A and B, and C and D, and the height was taken as the average of the sum of heights of A and D, and B and C. In these blurrier measurements there was more discrepancy between the x and y scales, up to 0.0022mm/pix. This was attributed to my eye's bias in choosing where line edges were, or to uneven distortion when not focused in the x and y directions. Currently, slope and intercept are 0.00010776, 0.01203661. The uncertainties and scatter of the points is large and this is attributed to it being very difficult to pick out the boxes on the very out of focus images. Because the scale is observed to be roughly monotonic, the validity of the positions and scaling of the photos taken was assumed to be unaffected by the fact that the camera was out of focus.

3.3 File List

Folder 1: MCP data

trinat@trinatdaq:/home/trinat/anholm/new_analysis/ncal7/rootplots/
Contains root files with names respective to run numbers

Folder 2: MCP data 2D histogram conversion

trinat@trinatdaq:/home/trinat/cpreston/march6_root2Dhistconvert/
cp_convertTH2_march6.com* This was altered as a batch script and run to convert the root files to 2D histograms

Folder 3: Main scripts that are involved in reading in camera and MCP data and comparing them

trinat@trcomp:/home/trinat/cpreston/analysis/april16_calib_fixverticalinvert/
plottogether4.m Depending on what runs are commented out and whether horizontal or vertical axis is selected, this is the main script that runs all the other scripts dependent on pathnames, but the pathnames are relative to my original cpreston@trcomp, so the path may have to be adjusted in this file to run on trinat@trcomp.
testplot5.m This is run for each trial and plots two sets of data centered
readin3.m Reads in MCP data
plotthem3.m Sets bounds for MCP data and prepares data
***.hist** These are the 2D histograms containing the MCP data extracted from ROOT files also in this folder, from Melissa's new calibration done March 17, 2015

Folder 4: Camera images and measurements

trinat@trcomp:/home/trinat/cpreston/analysis/feb20_TrapPosTest/
***.jpg**
Images taken for runs, with name of runs in title
matlabPosInfo.m/matlabPosInfo_inc767.m Contains horizontal edge information and measurements for centering, second file includes run 767
matlabPosInfoVert2.m/matlabPosInfoVert2_inc767.m Contains vertical edge information and measurements for centering, second file includes run 767
readin.m Reads in image files

Folder 5: Hz/Time plot information from ROOT MCP files

trinat@trcomp:/home/trinat/cpreston/analysis/march6_root2DHistconversion/

time_rateHz.m contains information from Frequency vs Time plot from ROOT files for MCP data - used for approximate normalization

4 Real-Time Control with Raspberry Pi

In upgrading the trap control system, there was interest in using a Raspberry Pi computer to replace the current DOS system. Benefits of this system include that it runs Raspbian, based on Debian, and hence provides the familiarity and convenience of a Linux system, including access to higher level languages and libraries. It has 40 configurable GPIO (General Purpose Input Output) pins that can easily be used to send and receive logic signals to control and read information from the experimental control devices including lasers, AOM's (acoustic optical modulators), DAC's, and cameras. It is also fast, with the Pi B+ model clocking in at 700MHz with a single core, and the Pi 2 model clocking in at 900MHz with its 4 cores.

4.1 Making Linux Real-Time

One very important factor was being able to configure the multitasking Linux system to reliably perform operations with real-time accuracy. This is important for a specific optical pumping event but also for the entire experimental control sequence. A real-time system performs operations with accurate and predictable timing. This type of system is important where failure of accurate timing could be disastrous, such as in machine operation or car electronic hydraulic systems. For the atom trap, latencies, or delays between trap events of tens of millseconds could be disastrous, allowing the trap to expand such that the location of the atoms are no longer well-known. In addition to certain custom-built hardware systems, Arduino and other microcontroller boards are also real-time systems, as they are dedicated to only run a particular sequence of tasks.

Though many computers seem blazingly fast and responsive, most computers are not real-time systems. This means that running processes can be interrupted and delayed, or preempted, by other more important processes without limits. This can cause delays of hundreds of milliseconds or longer. Popular computer operating systems including Windows, Linux, and Mac OS X are complex and designed for multitasking, or running many processes at the same time. They are optimized for overall system efficiency instead of the reliable execution of a single process.

On Linux, performance can be improved by giving specific processes higher priority than other processes. There are three types of scheduling policies that every process can be given on Linux. The `SCHED_FIFO` and `SCHED_RR` scheduling policies are "real-time" policies, and `SCHED_OTHER` is the normal scheduling policy for processes that don't need real-time capability. For `FIFO`, first in, first out, each process is run in the set order in the scheduler, and runs until it is blocked by an I/O request, or calls `sched_yield`. In `RR`, round robin, the runnable processes of a particular static, or assigned, priority are run for a specific period of time in turn, and then moved to the end of the execution queue for their specific priority. Static priorities for real-time policies can be set with a value between 0 to 99, with 99 being of highest priority. Processes scheduled with the default, normal `OTHER` policy can only be given a static priority of 0. The processor decides which process to run based on the list of processes on this schedule, giving them a dynamic priority, which can be influenced by the `nice` value. This `nice` value is a property of every process for this scheduling policy that influences its priority, but it is not its actual priority. Because the static priorities of real-time scheduling policies are higher, any real-time scheduled process preempts, or jumps ahead in line for execution, of any normal scheduled process. Dynamic priority values aren't calculated for the real-time scheduling policies, such that they are ensured to abide by their static priority and always be of higher priority than normal tasks. It should be noted that these real-time scheduling policies create soft real-time capability in Linux, which means that though the kernel tries to set scheduling priorities and deadlines, it will not always fulfill them. This alone does not achieve hard real-time capability that is required for our experiment [1] [2].

One major factor making it difficult to achieve hard real-time capability is that kernel processes, or those important to the core OS process that manages communication between software and hardware, immediately preempt all user processes. It operates in its own reserved portion of the computer, called "kernel space" where its most important processes have highest priority. There are a few ways to get around this. First, one can use the PREEMPT_RT patch for the Linux kernel, which makes most kernel processes preemptible, as we opted to do and is further discussed in the next section. Supposedly better timing can be achieved with systems, such as Xenomai, that run a real-time Xenomai operating system with Linux running as one of its processes. This makes it such that the Xenomai core handles real-time processes in interfacing with hardware while it also communicates with Linux that is operated by the user. One could also opt to try a completely non-Linux true real-time operating system - there are a few of these in development, such as Chibi OS. More information contrasting Xenomai and the RT patch can be found at [3].

4.2 Real-Time Kernel Patch

4.2.1 Description

There were a few reasons that we chose to try the PREEMPT_RT patch applied to the Linux kernel, to see if it would satisfy our real-time requirements. It would preserve the convenience of a full Linux system and our same coding structure with regard to communicating with hardware using C and C++ with Linux libraries. Considering that there were already packages available of Raspbian with the real-time patch applied, it seemed that it would prove the most convenient first option to try with our Raspberry Pi. With the prospect of exploring new, non-Linux territory if we were to venture to try Xenomai or another operating system, we thought it might be best to first try this available option, though it is my impression that the Xenomai system may also be not too hard to navigate with the Pi.

But what exactly does this PREEMPT_RT patch do? Mainly, it makes the majority of kernel processes preemptible, or able to be delayed or interrupted, by higher priority real-time user processes that would normally have to wait for the kernel processes to finish to execute. A few of these newly preemptible kernel elements include:

- Critical sections - areas of code that access a shared resource that can only be accessed by one process at a time
- Interrupt handlers - respond to signals from hardware
- Interrupt-disabling code - code that disables interrupt handlers
- Kernel locking - prevents access of same resource by two processes at once

It also gives priority inheritance to kernel locks, which means that if a resource is locked in the kernel by a low priority process, a high priority process that needs it can temporarily give the low priority process a higher priority just long enough to free the resource so it can be used by the higher priority process. It also does a few other things to remove unbounded latencies, or unlimited delays [4] [5] [6].

4.2.2 Installation

One reason we chose to try the PREEMPT_RT patch is that someone had already done the work to port it to work with the Raspbian kernel and compiled the kernel and created an SD card image. Seeing as I had had no experience with kernel patching and compiling, and this patch had to be specifically ported to work with this kernel version, we opted to try this version posted on the web by Emlid developers.

Emlid is a company that develops Navio+, a Linux-based autopilot that uses the Raspberry Pi. People can use these to make drones. Being an electronic robotic system, real-time operations are important for them, such that they have taken the trouble to port the real-time patch to the Raspbian kernel and offer their code to download on GitHub, as well as prepare SD card images that can be directly downloaded and loaded onto micro SD cards for use in the Pi. They have this available such that Navio+ customers can continue to develop the code according to their desires, and I have not noted any restrictions of usage or specified copyright.

The SD card image for kernel version 3.12 PREEMPT_RT and information for the Raspberry Pi A+/B+ is given at [7]. The 3.18 PREEMPT_RT version is given at [8] for the Raspberry Pi 2. These can be downloaded and directly written to micro SD cards, which can then be put directly into the Pi. It should be noted that in their kernel configuration and SD card image creation, Emlid removed a lot of programs that are included in the default Raspbian image, which may have to be installed separately. They also enabled and disabled certain functionalities as listed on their site to better promote connection with their Navio+ hardware. To write the images to the SD cards, I used both ImageWriter and the command line to do this, so far the result has been observed to be the same. This is documented in [9].

At one point I did try to download their code from GitHub [10] and follow their instructions to compile it for the Raspberry Pi B+, but it was different from the SD card image they provided, in that this was a kernel version 3.18 and not the 3.12 in the image, and I observed that it did not exhibit as good latency performance, pretty much on par with the original, non real-time kernel, according to Cyclictest. Since then I have only used their SD card images.

4.2.3 Testing With Cyclictest

Our preliminary testing was done in software with a program called Cyclictest. This is a common program used for benchmarking systems and obtaining general system latencies, or delays tasks experience depending on their priority. It has many test options that can be used to learn more about system delays. It creates a specified number of threads with the `SCHED_FIFO` real-time scheduling policy. Essentially, it measures latency of response to a stimulus event. In each thread, it loops a sequence where it gets the clock time, sleeps for an interval, then gets the clock time again, and the difference between the two clock times is calculated as the latency and over thousands of loops these latencies are plotted in a histogram. More information on Cyclictest can be found at [12] [13].

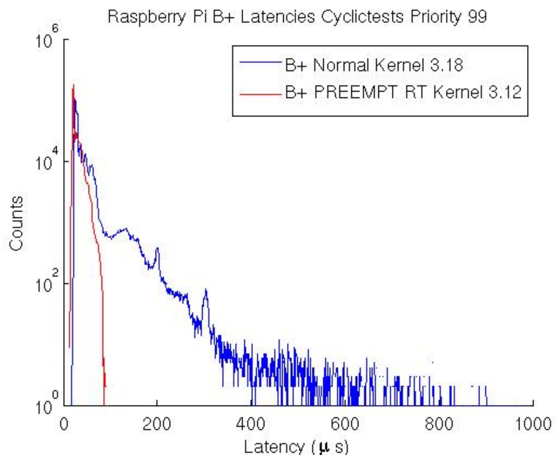
Cyclictest trials were run for each system:

- Raspberry Pi B+ normal PREEMPT 3.18 kernel
- Raspberry Pi B+ real-time PREEMPT_RT 3.12 Emlid-compiled kernel
- Raspberry Pi 2 normal PREEMPT 3.18 kernel
- Raspberry Pi 2 real-time PREEMPT_RT 3.18 Emlid-compiled kernel

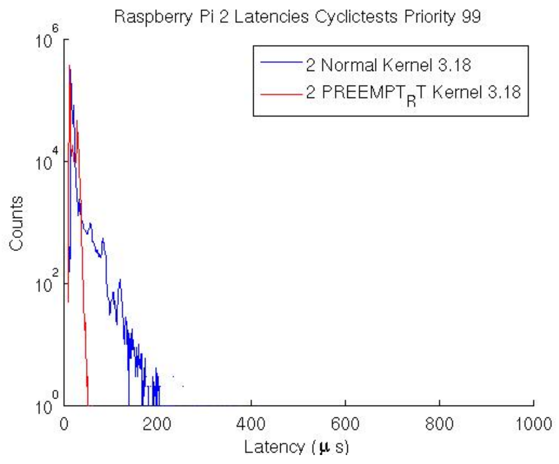
Results can be seen in Figures 4.1a and 4.1b. These results were obtained with the command `sudo cyclictest -l1000000 -m -n -a0 -t1 -p99 -i1000 -h1000`, creating one thread, looping 1 million times, real-time priority 99, creating a histogram with 1 ms as the highest bound for latency testing.

4.3 Programming

The priority and real-time scheduling policy for a process can be set in C code using `setpriority()`, and including `<sys/time.h>` and `<sys/resource.h>`. Other important actions to take to improve real-time performance include locking the process into memory, such that it can't be swapped out to the hard drive, which can slow down the process because writing to the hard



(a)



(b)

Figure 4.1

disk takes longer, using `mlockall()`. It is also a good idea to pre-fault the stack, a portion of memory that stores program variables, such that the program will know how much stack usage it requires at the start, such that it doesn't have to allocate more during execution, which could incur latency. This is all documented with example code on [4].

With a computer with multiple processors, such as the Raspberry Pi 2, one can change the boot settings such that processes will not be scheduled on a certain processor unless specifically requested. This allows us to dedicate one CPU entirely to our real-time process. This is done by adding `isolcpus=[cpu number]` to the GRUB boot loader configuration file and using CPU affinity to set the real-time task to execute on the isolated particular CPU. The last is done using `sched_setaffinity()` according to [11]. It is as of yet not very well known how much this improves the performance of the Pi 2 real-time processes, but it is known that for any processes of the same priority running on the same processor, they would have to compete with one another for processor usage and this prevents that. One can also control certain interrupt requests in having them only be sent to certain processors by changing the IRQ affinity. This was not really explored because the current system seems to have adequate timing.

It is important to note that setting a static priority of higher than 50, at which many kernel processes operate, will lock up the computer unless running the terminal and process on different processors. This was a problem for the B+ but not for the 2.

4.3.1 Programming with GPIO and DAC

In addition to the Cyclictest tests, we also needed to see how the system would perform with hardware. We used the GPIO pins as well as an ABElectronics DAC-ADC Pi that plugs directly onto the GPIO pins and communicates via SPI. It has 2 DAC's and 2 ADC's.

For SPI, the original ioctl functions included in native Linux libraries exhibited variant and frequent latencies of tens to hundreds of milliseconds. The wiringPi library was better, but best latency results were obtained with the bcm2835 library written by Mike McCauley [14]. It offers functions to access GPIO and other IO functions via the pins for the Broadcom BCM2835 chip on the B+, and can be easily updated to work with the BCM2836 chip on the Pi 2.

A few changes were made to this library to optimize performance and then recompiled as instructed on the website. All the $10\mu s$ delays in the SPI transfer functions were disabled to remove these unpredictable and unnecessary delays that incurred more latencies. The base peripheral address in the bcm2835.h file had to be changed from 0x20000000 to 0x3f000000 in changing from the B+ to the 2, because the Pi 2 has the bcm2836 processor, and this has a different base peripheral address [16]. The computer is supposed to find the address itself using device-tree, but this function was not working on the RT 3.18 kernel on the Pi 2, though device-tree seemed to be operating. This could be due to Emlid's kernel configuration. It should also be noted that the bcm2835.h file that came in the download was not updated with the RPi 2 GPIO pin addresses - this header file was looked up on the website [17] and these new pin addresses were added and the library recompiled.

The bit instructions sent to the MCP4822 DAC chip on the ABElectronics board were explored to learn how to control the gain. The bitwise sequence consisted of 16 bits total, with the first 4 being configuration bits, and the last 12 being the number value, as documented in [15]. Bit 15: selects channel A/B (0/1), Bit 14: unknown, left as default 1, Bit 13: Gain 1x, 2x (1/0), Bit 12: Shutdown signal (When =0, output of DAC shutdown), Bits 11-0: data bits.

Note that `memset(&[spi_ioc_transfer name],0,sizeof([spi_ioc_transfer name]))` had to be added to code prior to allocation of the bytes to send via SPI in order for the SPI calls to work. This it seemed cleared the buffers for sending and receiving.

Various sequences were programmed to be used to test latencies. The ones used to judge the real-time performance included the real-time implentation indicated. These included first, turning a GPIO pin on and off rapidly with no delays, and second, turning the DAC on and off rapidly with no delays with SPI. The speed was changed by adjusting the SPI clock frequency, and not adding times for the computer to sleep, to avoid incurring latency from this task. Another task used was a simulation of the specific laser sequence, of turning a GPIO pin on, turning the DAC to a different voltage, waiting 120 μs and then turning a GPIO pin off, and seeing how often and by how much this last event was late, relative to the pin turning on. In this the GPIO pin represented the laser in the laser cooling sequence, and the DAC changing the laser frequency. These were used to test the four systems, the RT and normal kernels for the B+ and 2 models.

Drift in timing as cycles continued can be minimized by using `clock_nanosleep`, with `CLOCK_MONOTONIC` to determine when next event should occur, instead of using relative sleep times [18] [19].

4.4 Latency Testing

Latencies were tested using an oscilloscope to observe output signals on scales large enough to see the sizes of latencies. These tests included watching the streaming signal and stopping the capture to observe the width of gaps in signals, as well as zooming in on a sequence and selecting the infinite persist option for the oscilloscope to see how the events behaved with respect to timing over many loops. Generally, latencies for all electronic tests were found to be characteristic to each system.

The B+ normal kernel was found to exhibit frequent latencies of up to hundreds of milliseconds and many far between. With the B+ real time kernel, this was reduced to a periodic latency of 50 milliseconds. The source of this was not identified, but likely a kernel process taking priority over our program, such as interrupt requests for the timer.

The Pi 2 normal kernel was found to exhibit latencies of a maximum of 60μs for 500000 events for a set of 4 instructions, while with the RT kernel this was reduced to a maximum of 30μs, making an approximate average of a maximum latency of 10μs per instruction following the first event. This was also run for longer, perhaps approximately 1000000 events, and no latencies above this were observed. Reasons this was thought to be better included that with four processors instead of just one, the processor could assign kernel processes and IRQ's to different processors such that they would not have to interrupt the process execution.

This maximum 30μs delay is one we believe satisfies the real-time requirements for our system control. Thus, we have decided to now use the Raspberry Pi 2 with the real-time kernel for our next stage of control development.

4.5 LabJack

We have had success in controlling a LabJack U3-HV board connected to the Raspberry Pi 2 running the RT kernel, intended for use in slower control. This has just been with using their sample C codes.

4.6 Power

The Pi has 1 A available to it from its power supply. From the GPIO pins the maximum draw should be 50mA, with no more than 16mA on a single pin. The digital signals were found to consistently meet the GPIO standard of 3.3V output. If powering a device from the 5V pins, it is connected directly to the USB charger, such that it can use any current that is leftover from what the Pi uses. Varying slightly for different models, the Pi may use up to 700mA maximum current, such that one might be able to power a device that needs 200-300mA to run. This does not account for extra USB devices connected that may draw more current. The Pi does not contain circuitry to protect it if supplied too much voltage or current through the pins or if shorted [20].

4.7 File List

Folder 1: Using DAC/GPIO on Raspberry Pi 2 RT (SD Card B2)

pi@trinatrpi2:/home/pi/abelecDAC/

dacalt2835PR2.cpp Using bcm2835 library, alternates DAC 1 between two voltage values at a set high real-time priority communicating via SPI. On April 8, 2015, fixed the dropping of the last byte. Contains functions for displaying voltage values and commands in binary

dacalt2835PRGPIO.cpp Using bcm2835 library, simulates atom trap laser cooling sequence, used to test latency in final event. Uses DAC and GPIO pin together, as well as `clock_nanosleep` delay, with ability to set high priority. Includes ability to set CPU affinity for 4 core system.

2835PRGPIO.cpp Uses bcm2835 library to test GPIO turning on/off rapidly to test its speed. CPU affinity has been added

2835PRGPIO_quitkey.cpp This is the same as above but with the added ability to press 'q' to exit the loop, the checking process in each loop takes approximately 2.75 μs

Folder 2: LabJack routine

pi@trinatrpi2:/home/pi/labjack/labjack-exodriver-136495f/examples/U3/

cp_u3allio.c This operates the Labjack U3-HV board with the Pi 2, by operating DAC 0 or 1 and reading the AIN's

4.8 Links

1. http://www.learnlinux.org.za/courses/build/internals/ch07s02.html
2. http://www.informit.com/articles/article.aspx?p=101760&seqNum=4
3. https://www.osadl.org/fileadmin/dam/rtlws/12/Brown.pdf.
4. https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO
5. http://www.linux.com/news/featured-blogs/200-libby-clark/710319-intro-to-real-time-linux-for-embedded-developers
6. http://lwn.net/Articles/146861
7. http://docs.emlid.com/Downloads/Real-time-Linux-RPi1/
8. http://docs.emlid.com/Downloads/Real-time-Linux-RPi2/
9. http://elinux.org/RPi_Easy_SD_Card_Setup
10. https://github.com/emlid/linux-rt-rpi

11. http://stackoverflow.com/questions/280909/cpu-affinity
12. http://events.linuxfoundation.org/sites/events/files/slides/cyclictest.pdf
13. https://mindlinux.wordpress.com/2013/10/25/using-and-understanding-the-real-time-cyclictest-benchmark-frank-rowand-sony/
14. http://www.airspayce.com/mikem/bcm2835/
15. http://tahmidmc.blogspot.ca/2014/10/pic32-spi-using-mcp4822-12-bit-serial.html
16. https://www.raspberrypi.org/forums/viewtopic.php?f=33&t=98740
17. http://www.airspayce.com/mikem/bcm2835/bcm2835_8h_source.html
18. http://stackoverflow.com/questions/7794955/why-is-clock-nanosleep-preffered-over-nanosleep-to-create-sleep-times-in-c
19. http://bec-systems.com/site/175/how-to-implement-realtime-periodic-tasks-in-linux-applications

20. http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/gpio-pin-electrical-specifications