Motion Control (at APS)

Most motion control is on APS experimental beamlines

Small amount on the accelerator side, mostly to control undulator gaps

Recent survey of beamline stepper motor controls (done for safety evaluation after a shock incident)

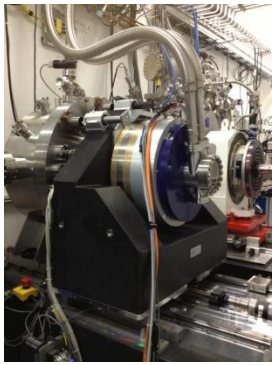
- 5,137 stepper motors
- 86 different models of motor drivers
- Most popular:
  - Advanced Control Systems Step-Pak (Unipolar, bipolar, mini-stepping bipolar)
- >25 different kinds of motor controllers (?)
- Most popular:
  - Pro-Dex OMS-58, MaxV (VME)
  - Delta-Tau Turbo-PMAC (Ethernet, VME)
  - Newport XPS (Ethernet)
  - Aerotech, Parker

APS Motor Software Support

- Most (but not all) APS beamlines run the EPICS control system
- Top-level object is the EPICS **motor record**
  - Lots of code has been written to this object:
    - spec, Python and IDL classes, etc.
  - "Least common denominator" support (acceleration, velocity, limits, etc.) but no advanced features
- 3 models of lower-level device and driver support have developed over time
- Original (Model 1)
  - Device-dependent device support and driver support for each controller type
  - Communication between device support and driver is custom for motor code and very limited
  - Cannot use other records to talk to driver, only motor record
  - Cannot take advantage of controller-specific features not supported by motor record
  - No provision for multi-axis coordination
  - Many EPICS drivers are written this way for historical reasons
- Model 2
  - Uses standard *asyn* interfaces to communicate between device support and driver
    - Implemented in C
  - *Can* use other records to talk to driver via asyn interfaces for controller-specific features
    - Not as easy as it should be to do so No implementation of multi-axis coordination

APS Motor Software Support

- Model 3
  - Two C++ base classes, asynMotorController and asynMotorAxis.
  - Base classes provide much functionality, only need to write device-specific implementations.
  - Easy to support controller-specific features
  - Don't have to use motor record.
  - Direct support for non-linear multi-axis coordinated "profile moves" in the driver API.
- Challenges
  - Improve efficiency of data collection by using only on-the-fly scanning
    - Upgrade drivers
    - Ancillary hardware (multi-channel scalers, detector trigger timing)
    - API abstraction needs to be proven on more hardware
    - Higher-level software support
  - Coordinate undulator motion with monochromators for on-the fly spectroscopy

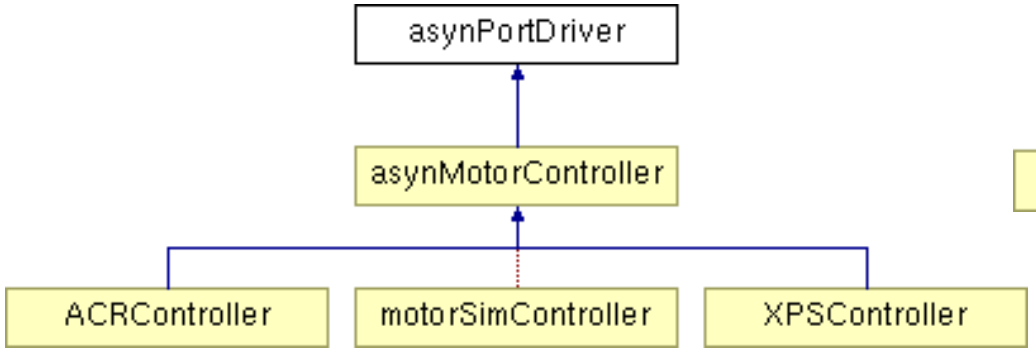


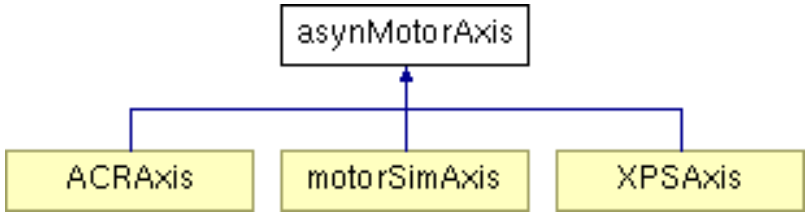
EPICS Motor Support

- Top-level object is the EPICS **motor record**
  - Lots of code has been written to this object:
    - spec, IDL and Python classes, etc.
- Next layer is EPICS **device support**
  - Knows about the motor record, talks to the driver
- Lowest layer is EPICS **driver**
  - Knows nothing about motor record, talks to the hardware
- 3 models of device and driver support have developed over time

Model 3

- Two base classes, asynMotorController and asynMotorAxis.
- Base classes provide much functionality, only need to write device-specific implementations.
- Easy to support controller-specific features
- Don’t have to use motor record.
- Direct support for coordinated profile moves in the driver API.





asynMotorController

```
class epicsShareFunc asynMotorController : public asynPortDriver {
  asynMotorController(const char *portName, int numAxes, int numParams,
                      int interfaceMask, int interruptMask,
                      int asynFlags, int autoConnect, int priority, int stackSize);

  virtual asynMotorAxis* getAxis(asynUser *pasynUser);
  virtual asynMotorAxis* getAxis(int axisNo);

  virtual asynStatus startPoller(double movingPollPeriod, double
     idlePollPeriod, int forcedFastPolls);
  virtual asynStatus wakeupPoller();
  virtual asynStatus poll();
  void asynMotorPoller();

  virtual asynStatus initializeProfile(size_t maxPoints);
  virtual asynStatus buildProfile();
  virtual asynStatus executeProfile();
  virtual asynStatus abortProfile();
  virtual asynStatus readbackProfile();
```

asynMotorAxis

```
class epicsShareFunc asynMotorAxis {
  virtual asynStatus move(double position, int relative,
            double minVelocity, double maxVelocity, double acceleration);
  virtual asynStatus moveVelocity(double minVelocity, double maxVelocity,
            double acceleration);
  virtual asynStatus home(double minVelocity, double maxVelocity,
            double acceleration, int forwards);
  virtual asynStatus stop(double acceleration);
  virtual asynStatus poll(bool *moving);
  virtual asynStatus setPosition(double position);

 virtual asynStatus initializeProfile(size_t maxPoints);
  virtual asynStatus defineProfile(double *positions, size_t numPoints);
  virtual asynStatus buildProfile();
  virtual asynStatus executeProfile();
  virtual asynStatus abortProfile();
  virtual asynStatus readbackProfile();
```

Coordinated Motion: Motivation

- Traditional step scanning overheads:
  - Start and stop motors & mechanical systems
  - Arm and read detectors.
- Typically much more efficient to collect data on-the-fly as the motors are moving.
  - Better detectors, more flux with APS Upgrade, don't waste photons and time
- However, on-the-fly scanning with EPICS has been subject to limitations.
  - Typically limited to simple single-motor scans because EPICS did not provide a way to program motor controllers to perform synchronized motion of multi-axis controllers.
    - Exception to this was the Trajectory Scanning software I wrote for the Newport XPS and MM4005 motor controllers.
    - Not a clean solution: EPICS SNL program talking directly to the controller, "behind the back" of the motor record driver.
    - PMAC does have motion control programs – good for fixed, simple geometry
  - New solution that incorporates a coordinated motion API in the motor driver layer

asynMotorController

```
class epicsShareFunc asynMotorController : public asynPortDriver {
  asynMotorController(const char *portName, int numAxes, int numParams,
                      int interfaceMask, int interruptMask,
                      int asynFlags, int autoConnect, int priority, int stackSize);

  virtual asynMotorAxis* getAxis(asynUser *pasynUser);
  virtual asynMotorAxis* getAxis(int axisNo);

  virtual asynStatus startPoller(double movingPollPeriod, double
     idlePollPeriod, int forcedFastPolls);
  virtual asynStatus wakeupPoller();
  virtual asynStatus poll();
  void asynMotorPoller();

  virtual asynStatus initializeProfile(size_t maxPoints);
  virtual asynStatus buildProfile();
  virtual asynStatus executeProfile();
  virtual asynStatus abortProfile();
  virtual asynStatus readbackProfile();
```

asynMotorAxis

```
class epicsShareFunc asynMotorAxis {
  virtual asynStatus move(double position, int relative,
              double minVelocity, double maxVelocity, double acceleration);
  virtual asynStatus moveVelocity(double minVelocity, double maxVelocity,
              double acceleration);
  virtual asynStatus home(double minVelocity, double maxVelocity,
              double acceleration, int forwards);
  virtual asynStatus stop(double acceleration);
  virtual asynStatus poll(bool *moving);
  virtual asynStatus setPosition(double position);

  virtual asynStatus initializeProfile(size_t maxPoints);
  virtual asynStatus defineProfile(double *positions, size_t numPoints);
  virtual asynStatus buildProfile();
  virtual asynStatus executeProfile();
  virtual asynStatus abortProfile();
  virtual asynStatus readbackProfile();
```

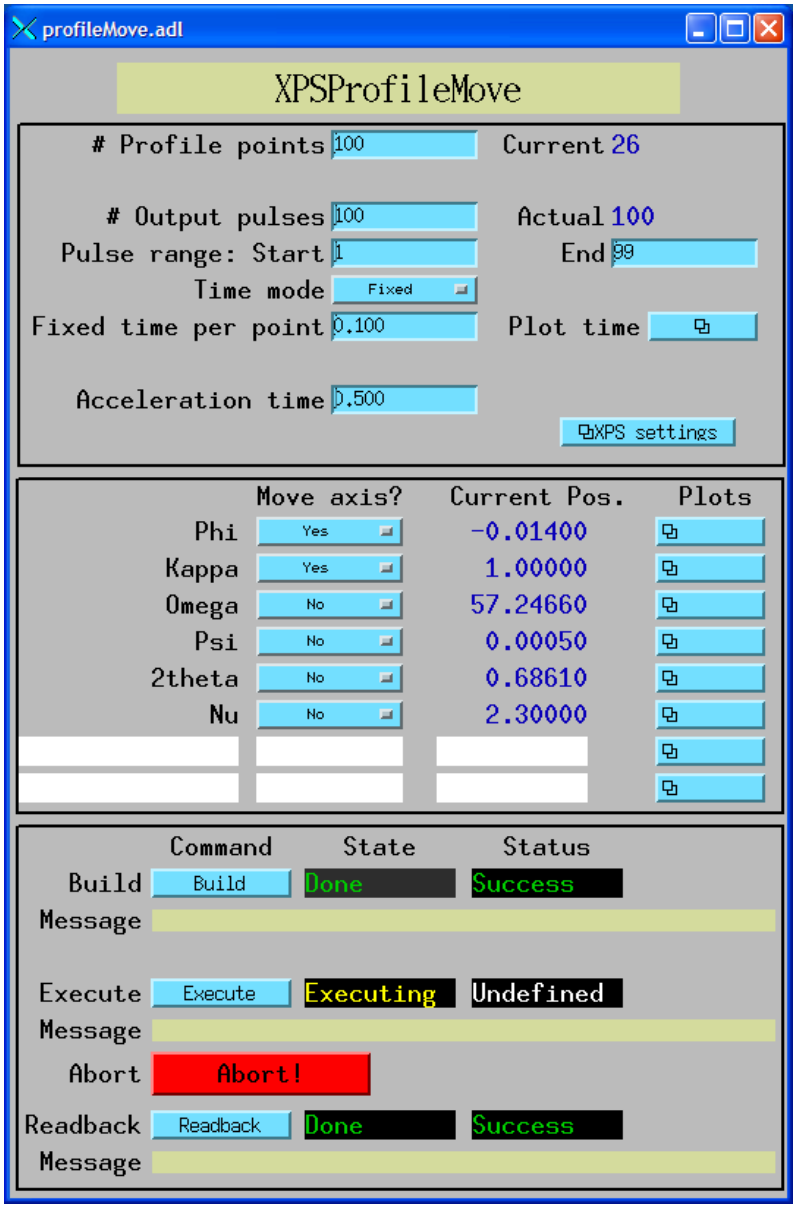
Coordinated Motion
(Now called ProfileMove, previously TrajectoryScan)

- Create arrays of target locations of each motor in a multi-axis controller
- Create array of time per element
- Define times/locations to output synchronization pulses to trigger detectors
- Execute coordinated move
- Optionally read back encoder positions when each synchronization pulse was output
- Many controllers have this Position/Velocity/Time (PVT) capability
  - Newport XPS, Delta Tau PMAC, Pro-Dex MAXv, Parker ACR series, Galil etc.
- We now have an API to take use this capability in a consistent way
- Now done in same driver as for motor record, eliminates conflicts

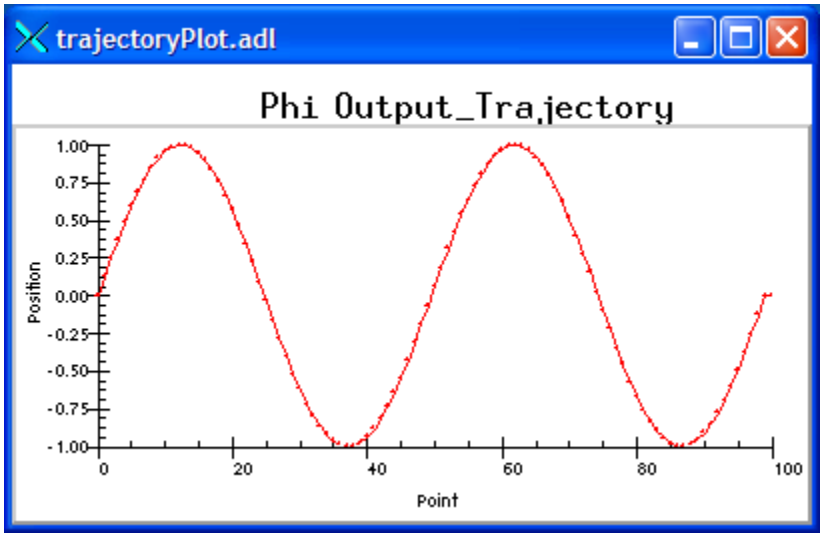
Client Software

- Interface from clients is simple
- Define arrays containing target location for each axis in each segment of scan
- Define array of time per point for each segment of scan
- Define when to output trigger pulses for detector
- Send these arrays to EPICS records
- Build and execute scan
- Optional read back actual encoder positions at each point in scan
- Easy to implement in any language with EPICS bindings:
  - Python, SPEC, Matlab, IDL, C/C++, etc.

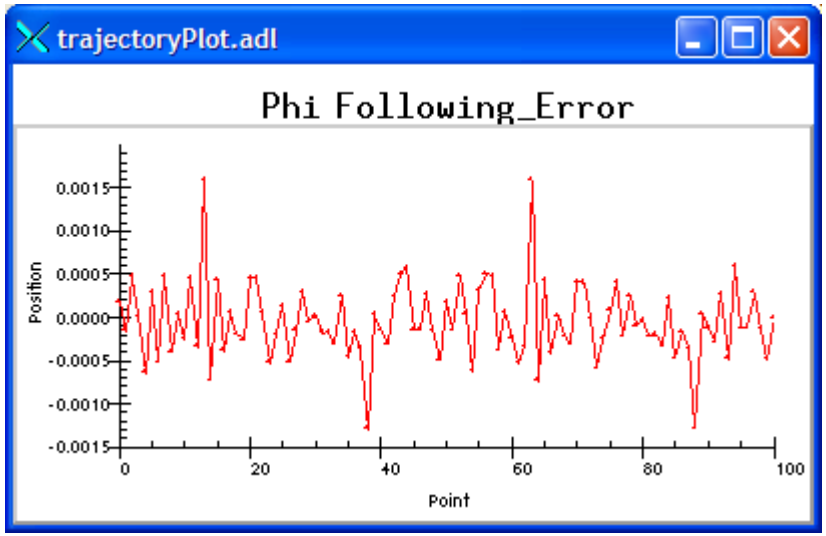
Coordinated Motion



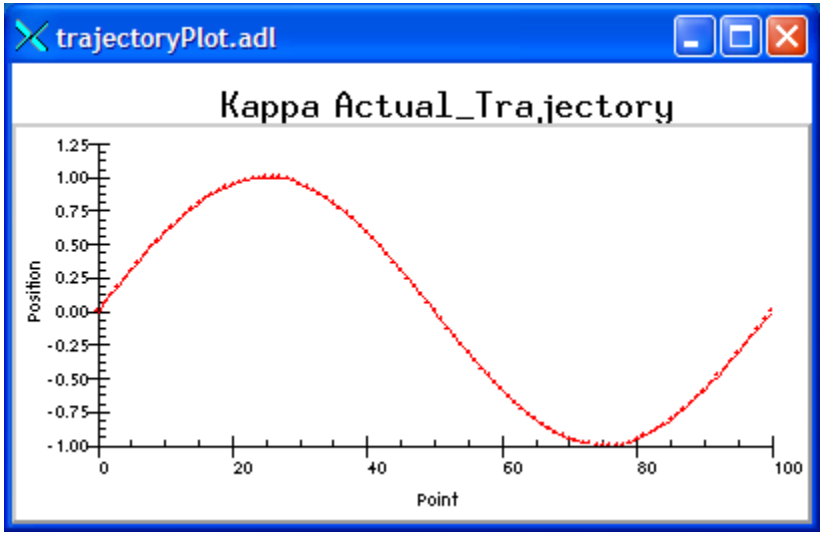

```
; IDL program to build, execute and readback profile
; This program builds a profile and executes it.

nelements = 100 ; 100 elements in the profile.
npulses = 100 ; 100 pulses in the profile.
naxes=2       ; We will move the first 2 motors (Phi and Kappa)

; Define array of positions
positions = dblarr(nelements, naxes)

; The Phi profile is a sin wave with two complete periods and an
; amplitude of +-1 degrees
positions[*,0] = 1.*sin(findgen(nelements)/(nelements-1.)*4.*!pi)

; The Kappa profile is a sin wave with one complete period and an
; amplitude of +-1 degrees
positions[*,1] = 1.*sin(findgen(nelements)/(nelements-1.)*2.*!pi)
profile = 'IOC:Prof1:'
group = 'GROUP1'

time = 0.1      ; Fixed time per profile point
status = profile_move(profile, positions, group=group, maxAxes=6, $
            build=1, execute=1, readback=1, time=time, $
            npulses=npulses, actual=actual, errors=errors)
end
```



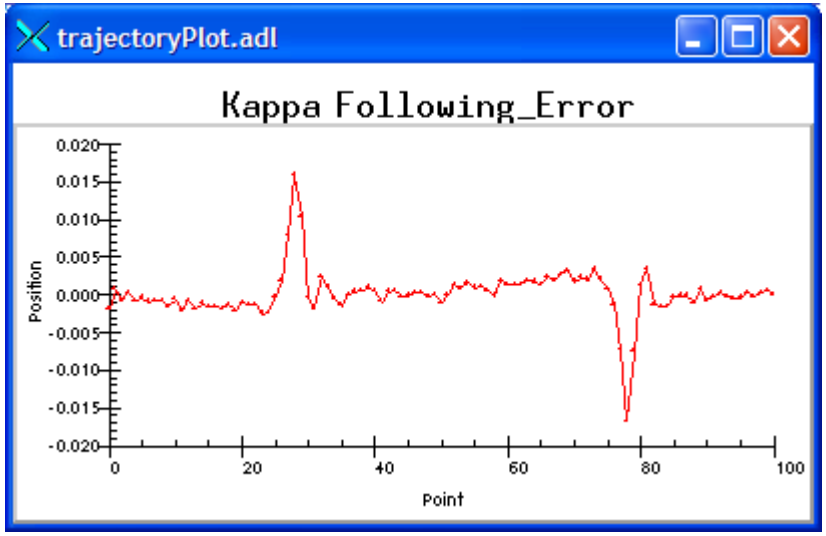
Phi output profile



Phi following error



Kappa readback profile

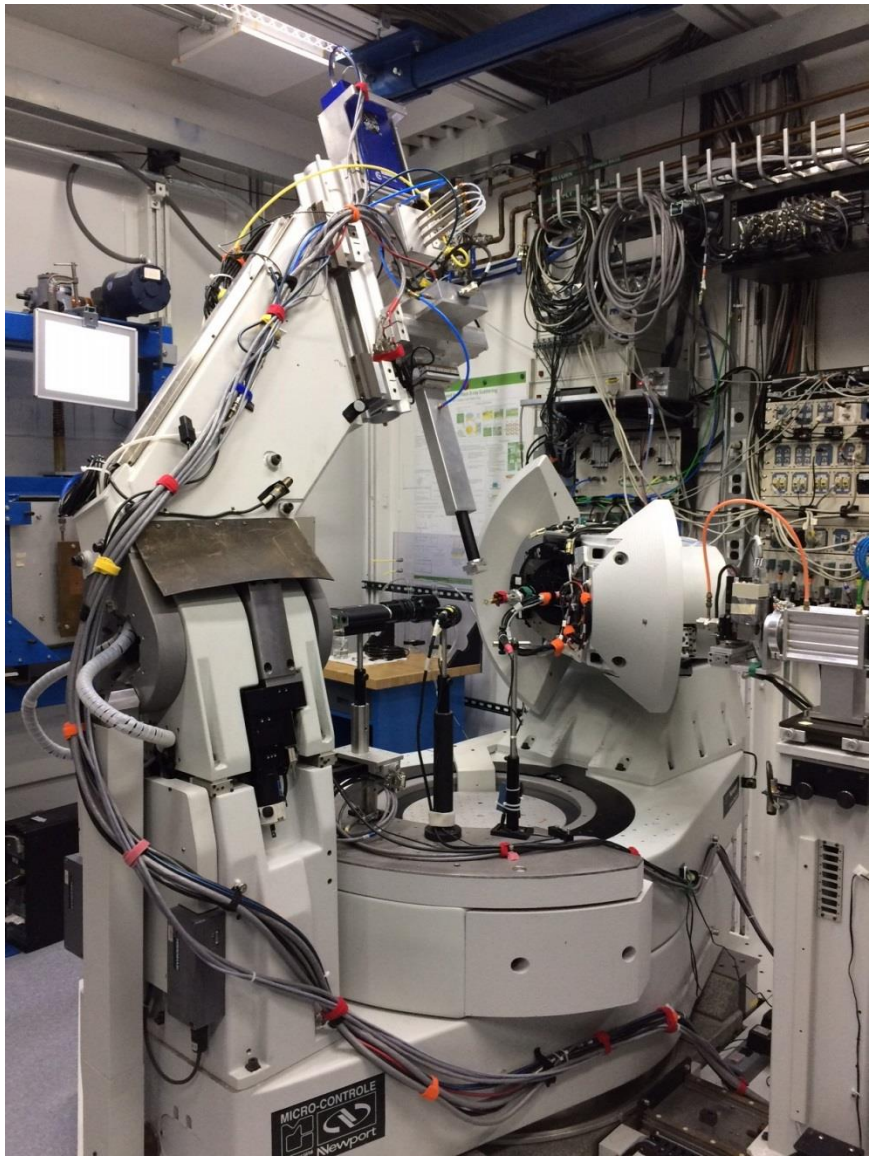


Kappa following error

SPEC Interface

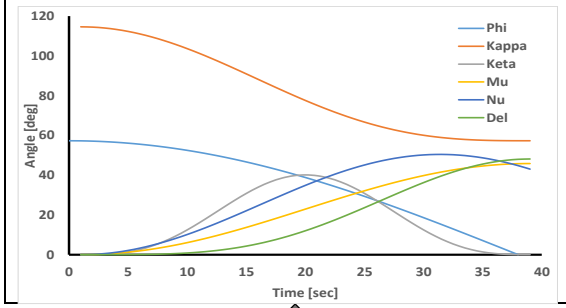
- SPEC macros have been written to allow SPEC to utilize profile moves via EPICS interface
- Low level interface, all of SPEC's standard scans can be done "on-the-fly" with profile move software. Replacement macros for:
  - _ascan # Used by all ascan and dscan macros
  - Mesh
  - hklscan # Used by hscan, kscan and lscan
  - _hklmesh
  - _hkllline # Used by hkcircle, hlcircle, klcircle, hkradial, hlradial and klradial
  - _scanabort resume
  - _loop

SPEC Scan Using Newport 6-Circle Diffractometer



- Newport XPS is used to drive 6-circle diffractometers at APS (3), SLS
- SPEC is the control software
- Use Profile Moves both for initial alignment and for data collection
- Pilatus 100K is typically used for collecting crystal truncation rod data
- Up to 200 frames/s
- Use counts in areaDetector ROI and statistics plugins as a SPEC counter
- Pilatus TIFF files are saved for use with later data analysis

SPEC 6-Circle Diffractometer



EPICS Trajectory Scanning Support

Newport XPS

Complex Coordinated Trajectory Using PVT

Record Encoder Values at each Trigger

Struck Scaler

Trigger

Trigger

I

o

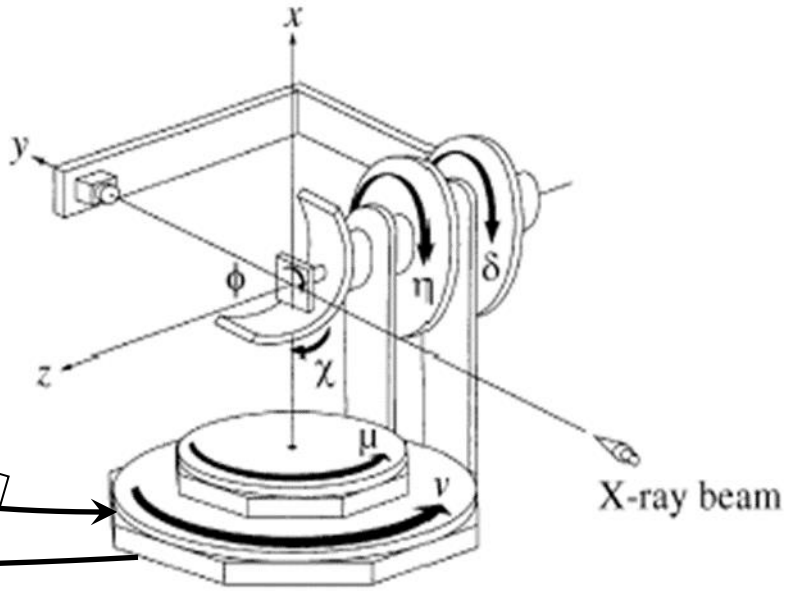
I_1

Pixel Array Detector 100k Pilatus

Readout time	**2.7 ms**
Framing rate	**200 Hz**
Dimensions	**275 x 146 x 85 mm**

DC Servo Control

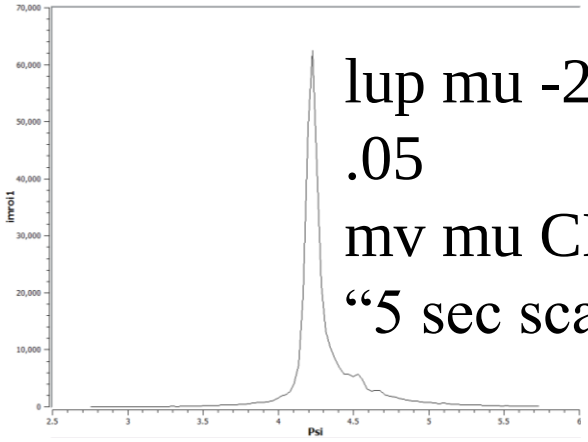
Encoder Read-Back for Data Logging



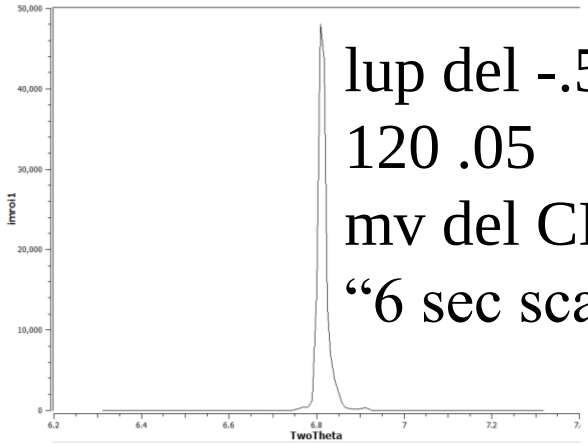
Newport 6-Circle Diffractometer

Trajectory Scan For Single Crystal Alignment

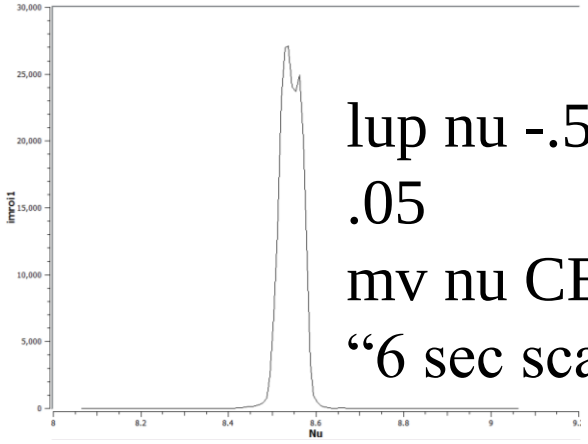
1) Scan Mu to maximize Bragg peak on Ewald Sphere
2) Scan Del to horizontally center Bragg peak on detector
3) Scan Nu to vertically center Bragg peak on detector



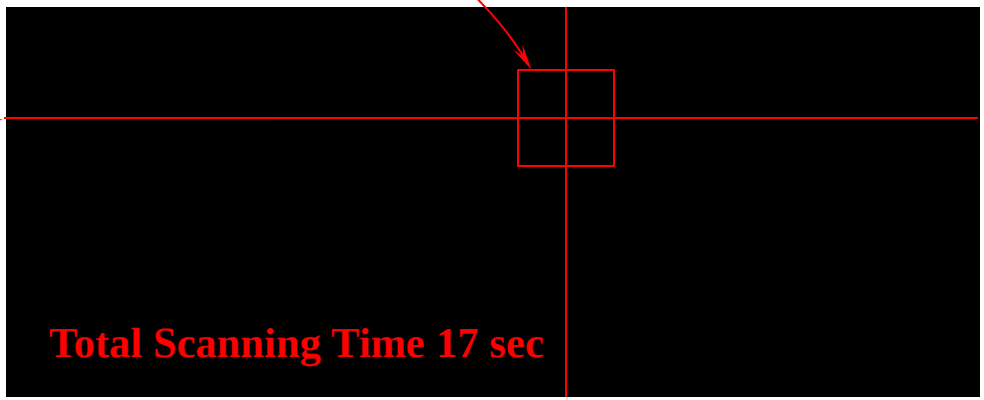
lup mu -2 2 120 .05
mv mu CEN
“5 sec scan”



lup del -.5 .5 120 .05
mv del CEN
“6 sec scan”

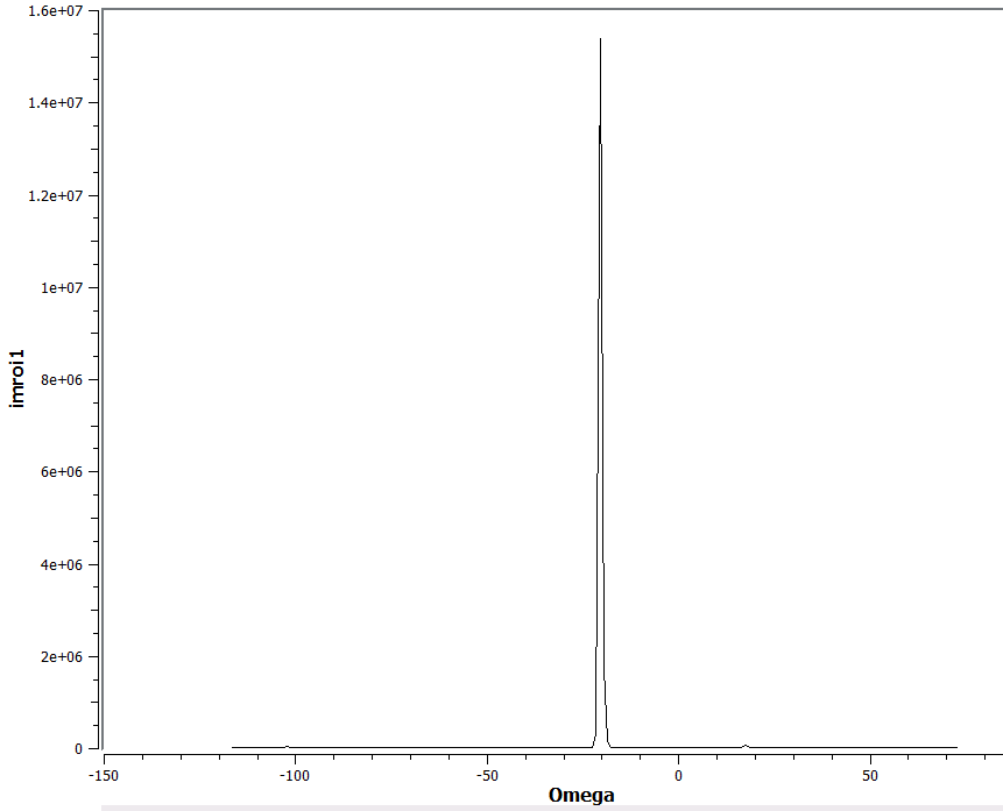


lup nu -.5 .5 120 .05
mv nu CEN
“6 sec scan”



Pilatus 100 k Pixel Array Detector

- **Fast wide scan to find first reflection**
- 190 deg Scan in Omega, 47 sec total scan time
- On The Fly – the detector is always measuring so the sharp Bragg peak is not missed

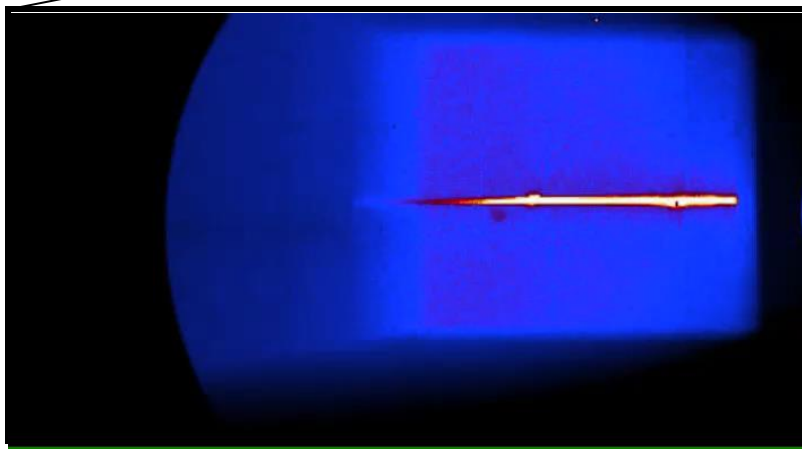


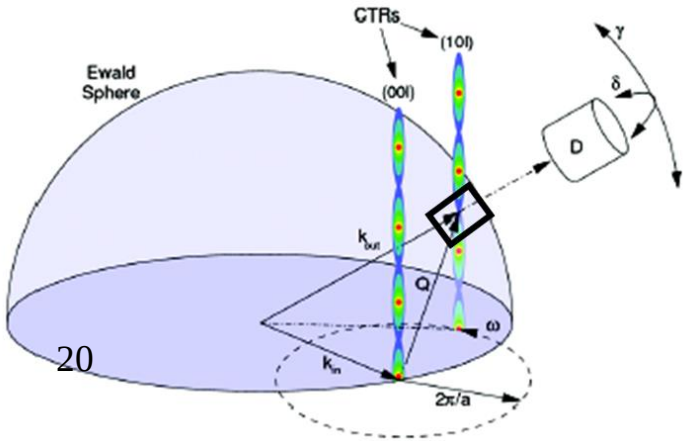
Automated single crystal orientation refinement using trajectory scans to center on each Bragg peak

```
br 0 0 12
do HKL_lup.mac
orient_add 0 0 12
or0 0 0 12

# 2
br 1 0 4
do HKL_lup.mac
orient_add 1 0 4
or1 1 0 4

# 3
br 2 -1 3
do HKL_lup.mac
orient_add 2 -1 3

# 4
br 1 -2 6
do HKL_lup.mac
orient_add 1 -2 6

# 5
br -2 -1 8
do HKL_lup.mac
orient_add -2 -1 8

# 6
br -1 0 5
do HKL_lup.mac
orient_add -1 0 5

# 7
br -2 1 6
do HKL_lup.mac
orient_add -2 1 6

# 8
br -1 3 8
do HKL_lup.mac
orient_add -1 3 8

# 9
br 0 2 4
do HKL_lup.mac
orient_add 0 2 4

# 10
br 1  1 6
do HKL_lup.mac
orient_add 1  1 6

orient_show
print "done with orienting the crystal"
```

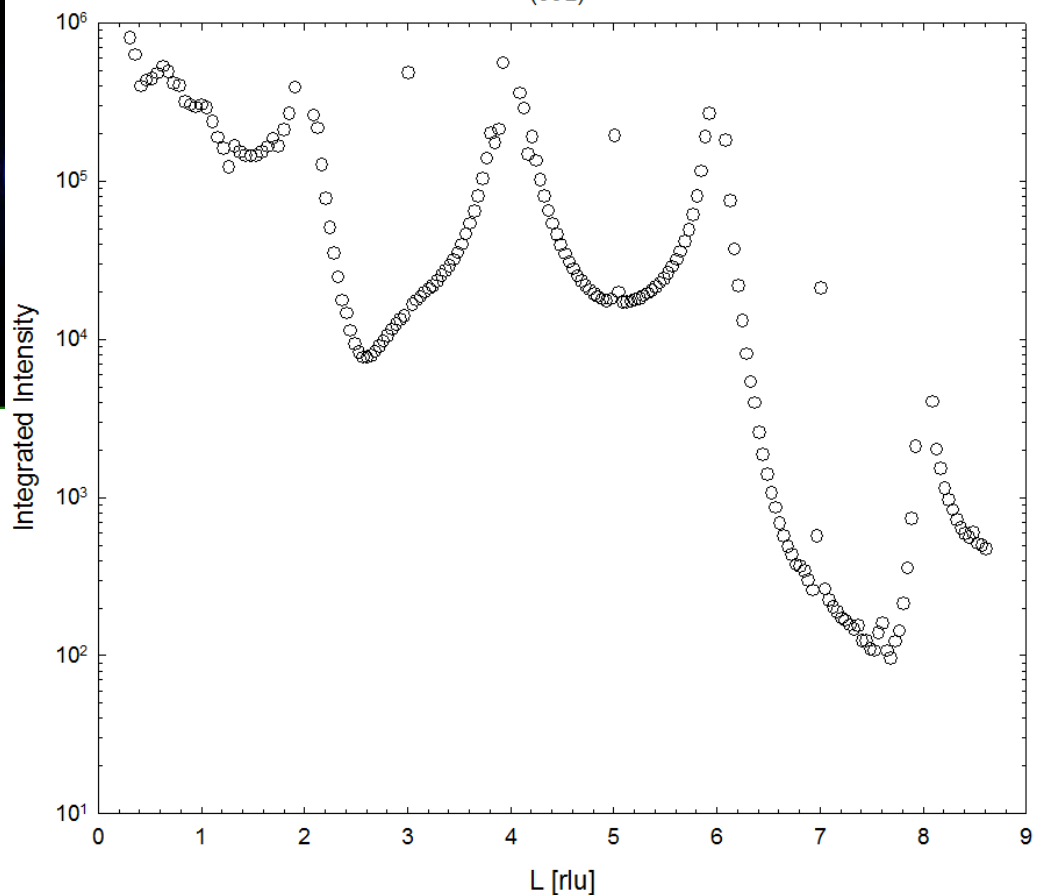
Crystal Truncation Rod (CTR) measurement using complex coordinated motion control and on-the-fly data collection

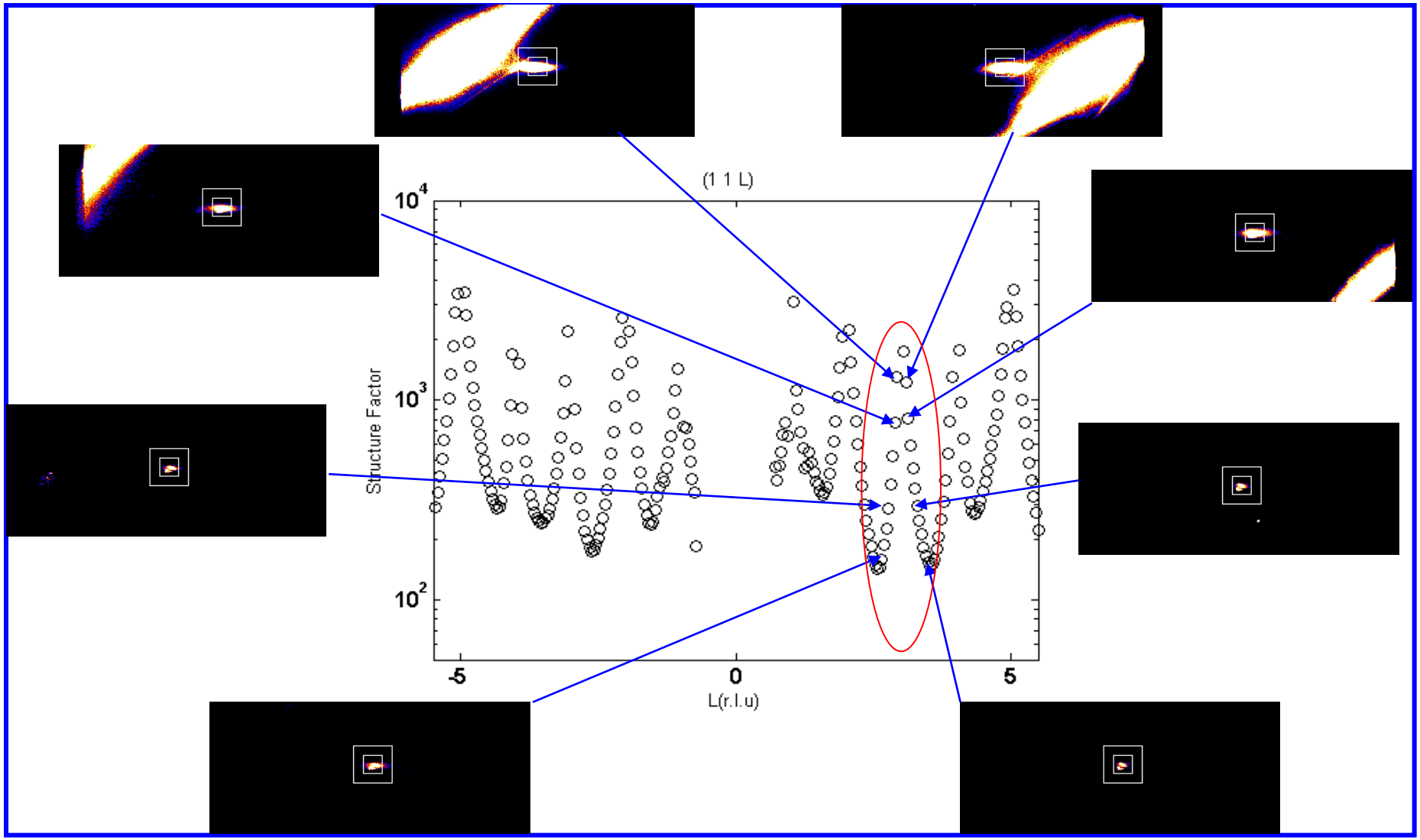
FeOOH 00L Rod





Total Scan time 38 min




(1 1 L)
Structure Factor
10^4
10^3
10^2
-5
0
5
L(r.l.u)

Future Work

- Convert Pro-Dex MAXv, Aerotech Ensemble to Model 3
  - Tim Mooney already has trajectory scanning running in SNL program on each of these
- Support other controllers
- See if API needs changes based on mapping to other controllers
- Work on replacement for EPICS motor record?