Development of computer vision and image processing libraries at the National Synchrotron Light Source - II

William Watson

Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218

Kazimierz Gofron

Department of Photon Sciences, Brookhaven National Laboratory, Upton, NY 11973

Abstract

The beamline microscopes at the National Synchrotron Light Source – II (NSLS-II) at Brookhaven National Laboratory utilize X-ray imaging to enable the study of material properties and functions with nanoscale resolution. Along these beamlines are hutches that contain detectors and cameras to monitor the state of the beam and samples. Since the hutch must be sealed during operation, robots help to automate the process of sample transfers and camera movement without human interference. However, in order to automate these processes, the development of computer vision and image analysis software is required. By building a general purpose computer vision module with the support of OpenCV, an open-source computer vision library, computers now have the ability to determine if a robot has improperly mounted a sample or pin, and can warn users and scientists of potential errors within the hutches. This wrapper module allows users to simplify computer vision processes and programs with succinct code. Computer vision also allows computers to determine the positon, spread, and intensity of the x-ray beams automatically from images provided by cameras on the beamline. By integrating computer vision to the Experimental Physics and Industrial Control System (EPICS), an open source control system for scientific instruments, computers can, in real-time, observe and report of the status of a beamline without human interference or control. This will help prevent improper mounting of samples and possible collisions that will result in downtime and damage to the beamline. This will also allow for automated error checking, beam analysis, and robotic experimentation without input from beamline scientists. The library has potential for more applications to other beamlines. As a result, I have added computer vision to my repertoire of computer techniques. Additionally, I am now familiar with EPICS and the OpenCV library implemented in the Python programming language.

I. Introduction

The development of computer vision and image analysis software at NSLS-II will provide beamline scientists and users the tools needed to interpret data and streamline experimental processes automatically from camera images. The development of an easy to use python module will allow controls engineers and scientists to quickly develop smaller applications that process camera images efficiently and make experimental decisions based upon the data collected. This will allow beamline scientists, controls engineers, and users to automate tedious and costly processes. Applications will be able to determine the position, spread, and intensity of X-Ray beams. Applications will be able to isolate and compute the centroid, or center of mass, of features within the images. Finally, programs will be able to provide assistance to users and scientists in the alignment of goniostats and samples within the hutches of the various beamlines.

An easy, user-friendly python module implements many OpenCV functions, along with the appropriate pre-processing and additional algorithms, to allow for the development of succinct and elegant computer vision code. The python module, currently known as cvlib, allows for small programs to quickly delegate processes for the purposes of image processing. It has significantly reduced the amount of code required of programmers to write in order to implement certain procedures. This library implements a software design technique known as modular programming, emphasizing the separation of tasks and functions, allowing for the construction of large programs with minimal waste. Many functions take optional arguments, allowing for inexperienced users to develop quick flexible applications, while the more advanced will have unfettered access to the minute details of each function.

OpenCV is an open source computer vision library that was designed for computational efficiency. It is written and optimized in C/C++ code, resulting in some of the fastest

computation that a high-level programming language can provide.[1] As of OpenCV version 3.1.0, the library implements the Open Computing Language (OpenCL), allowing for algorithms to make the most of hardware acceleration in the underlying computer platform. OpenCV includes licenses to Intel's Integrated Performance Primitives (IPP) library, as well as the Intel Threading Building Blocks (TBB) library.

The Intel IPP library is a crucial package necessary for the computational efficiency of OpenCV. Integrated heavily within the functions of OpenCV, the Intel IPP library, implemented in C/C++, allows for applications to take full advantage of low-level processing power, speeding up computation times for various functions by factors of three to nine.[2] This results in a low average computational cost for images, allowing for real-time analysis of images on the beamlines. OpenCV makes extensive use of the Intel TBB library, which allows for parallel programming on multi-core processors.[3] TBB allows for applications to create synchronized tasks that it delegates to low-level processors, allowing tasks to be completed in parallel instead of sequentially, drastically improving the computation time needed for real-time analysis.

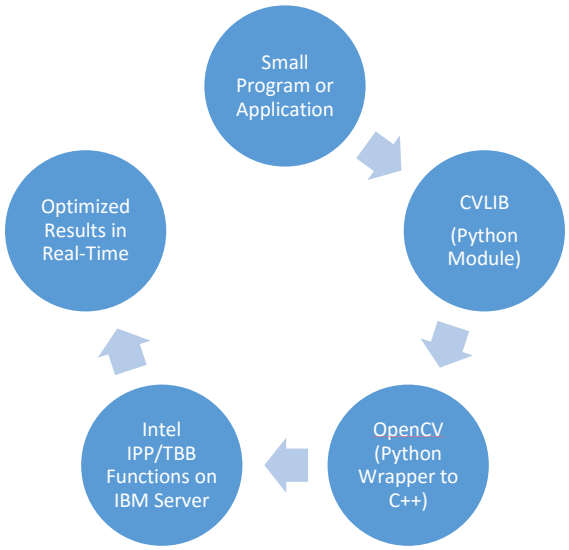

Figure 1: Infrastructure Layout for Computer Vision

This extensive infrastructure is hosted on a dedicated IBM Image Server that is built and optimized for computational speed and efficiency, providing near instantaneous results to users. Debian 'Jessie', the current stable release of a powerful Linux distribution, was installed on the IBM Image Server to replace the old operating system. With the infrastructure running locally on

the beamline, computer vision applications can be integrated with the Experimental Physics and Industrial Control System (EPICS). EPICS is an open source control system for scientific instruments, allowing beamlines to store data from all the instruments and devices along the beamline in one convenient database.[4] By accessing these values from Process Variables, or PV values, computer scripts can adjust motor and instrument controls quickly, while retrieving image data directly from cameras. By allowing scripts to adjust these parameters, computer vision can be used to determine the positon of samples or mounts within an image and automatically adjust the motors to correctly calibrate the beamline. The use of the computer vision module to directly send motor adjustment to calibrate beam positon or correct goniostat alignments can eliminate manual calibration. By integrating this module and OpenCV with EPICS, engineers will be able to develop robotic vision for the beamlines.

II. Computer Vision Applications and Results

A. BPM-1 Camera (IXS)

At the Inelastic X-Ray Scattering (IXS) Beamline at NSLS-II, cameras are used to record and interpret data received from the hutches regarding the shape and intensity of the beam. One particular example is the BPM-1 X-Ray camera located in B Hutch at IXS. This camera captures X-Ray images as it passes through the beamline.

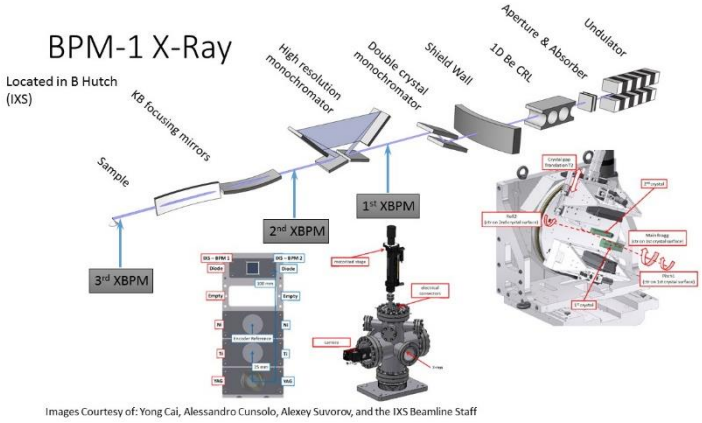

Figure 2: IXS Beamline Schematic[5]

The X-Ray beam originates in the undulator, where it passes through various double and high resolution monochromator crystals to focus the beam into a monochromatic x-ray. Before it hits the sample and continues to the end station in D Hutch, cameras record the properties of the beams. As the beam passes through the double crystal monochomator, it is partially reflected by a YAG crystal inside a scintillator, allowing a replica of the beam to be passed to a camera, in this case BPM-1. The beam is oval because it is partially reflected by the scintillator. If the camera was position directly in front of the beam, it would be circular instead of elliptical.

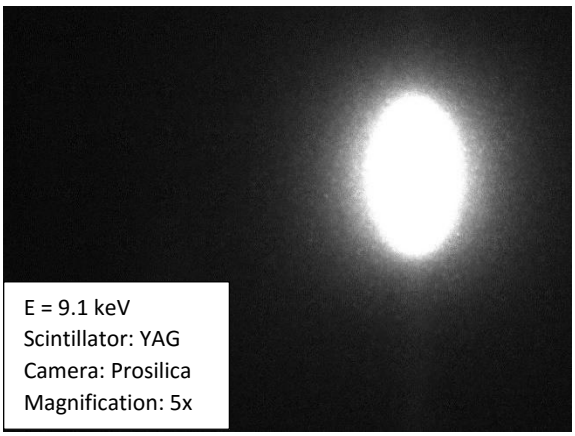

Figure 3: BPM-1 Captured Image

The application developed for the BPM-1 camera allows beamline scientists to quickly discover the position, spread, intensity, and centroid of the beam. As the images are captured from the camera, a simple three step process is used to retrieve the pertinent information. As images are loaded into the application, a simple binary threshold is applied, resulting in a binary image. A binary image is an image where only two values are present, high and low. In this case, high corresponds to white (255) and low corresponds to black (0). For this program, the

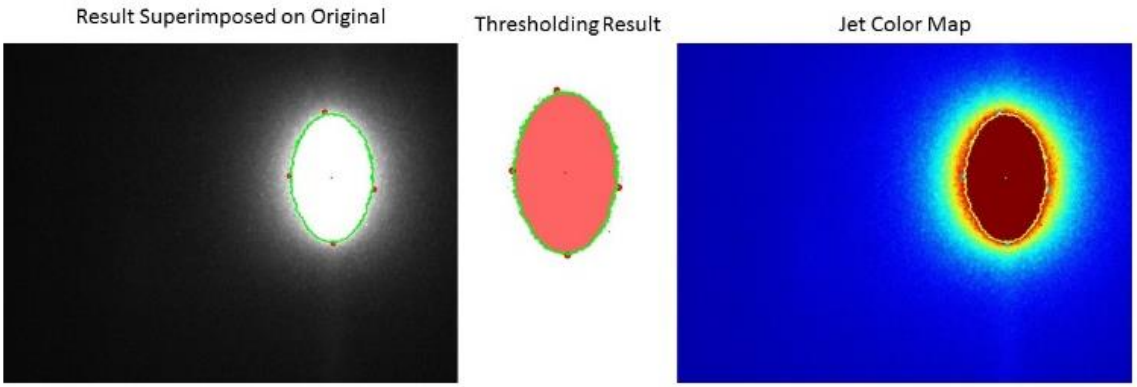

Figure 4: BPM-1 Image Results

threshold is set at 220, in order to only retrieve the pure and intense white pixels. Then, the findContours() method is applied, returning all contours in the image by size, such that the largest contour is the first item in the list. Since the beam is always the largest object, the first list of points corresponds to the beam's contour. The contour only surrounds the intense, bright white pixels. The resulting image displays the contour, the centroid, and the four extrema of the contour found.

With the contour, centroid, and extrema found, the program outputs relevant information of the beam, such as the perimeter, orientation, maximum point, height, area, minimum point, sum of pixels, width, and mean intensity. From these values, beamline scientists can easily discover the beam's intensity from the sum of the pixel values, while knowing the spread of the beam from the height and width values. By using these values as parameters, beamline scientist can calculate the offset of the beam to the center of the image frame and recalibrate the beam as needed. With EPICS integration, beam alignment can

Figure 5: Console Output

```
Object Details:
perimeter: 2356.99022925
orientation: 179.838363647
max: (925, 198)
height: 372
extrema: {'B': (938, 568), 'R': (1054, 415),
          'L': (813, 377), 'T': (914, 196)}
area: 65058.5
min: (1047, 564)
sum intensity: 20426526
width: 241
centroid: (933, 382)
mean intensity: 227.842390577
```

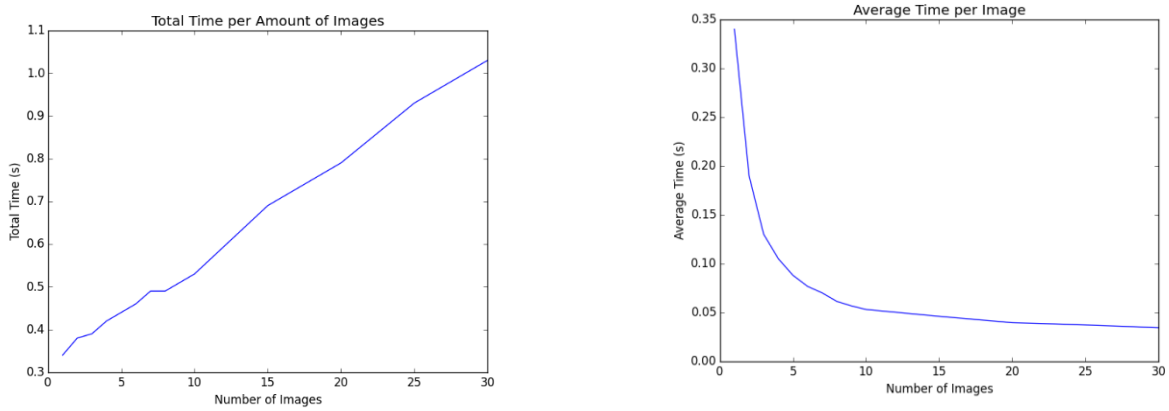

Figure 6: Total Time Chart (Left) and Average Time Chart (Right)

happened instantaneously without the need for human input. As a result, pertinent information and beam alignment can be easily done in real-time.

Due to the overhead that comes with python, a single image takes approximately 0.350 seconds to calculate. The average time per image can be reduced to approximately 0.034 seconds, allowing for the program to process 30 images per second if need be.

B. Merlin Quad X-Ray Detector (IXS)

The Merlin Detector in D Hutch at IXS collects information regarding X-Rays. Using direct detection, the Merlin Detector collects X-Ray intensities after the X-Ray passes through six crystals that divide the beam into six streaks. At the time of testing this program, IXS only configured five streaks to be monitored, as the images were collected during initial calibration.

The purpose of this program was to isolate the individual streaks found within the

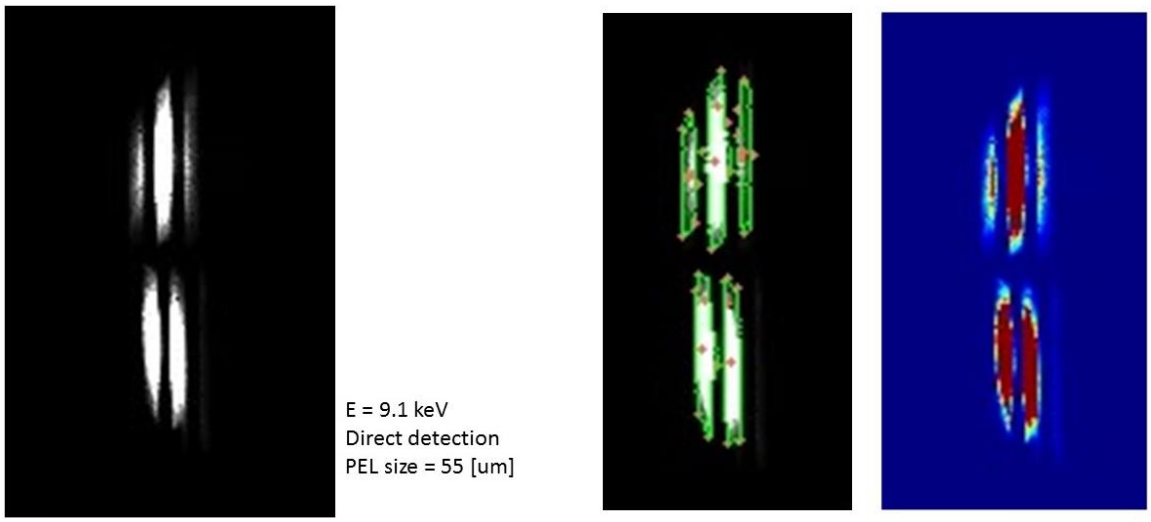

Figure 7: Console Output for the largest X-Ray Streak (Blue Rectangle)

collected images. When isolated, the program will process each streak to discover the position, center, spread, intensity, and the maximum value. By passing the image to the program, a simple binary threshold was applied, enabling OpenCV to find the contours in order of size. By enumerating through the list of contours, filtered for small noise interference, the program can

extract data for each streak and record it efficiently. The Merlin Detector collected information

Console Output:

```
Object 1:
perimeter: 125.840619564
orientation: 179.981033325
max: (131, 78)
height: 55
extrema: {'B': (129, 122), 'R': (135, 98),
          'L': (126, 92), 'T': (132, 67)}
area: 270.5
min: (134, 83)
sum intensity: 62689
width: 9
centroid: (130, 95)
mean intensity: 126.644444444
```

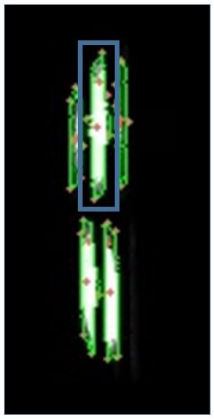

Figure 8: Merlin Image Results

on five streaks in the input image, and correctly discovered the contours in the resulting image on the right. A JET color map is provided to accentuate the pixel values. The program can return the information for all streaks within a Merlin image in approximately 0.321 seconds for one image. When multiple images are sent in sequentially, being processed by one instance of the program, it can process approximately 30 images per second. The python overhead for compiling and garbage collection becomes negligible upon a large number of images being process by one instance of the program.

C. On Axis Pin Locator and Alignment (IXS)

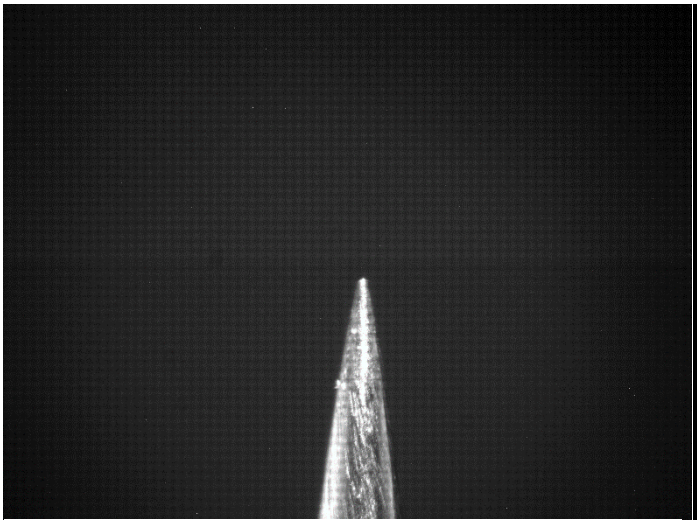

Figure 9: On-Axis Pin Image Captured from IXS

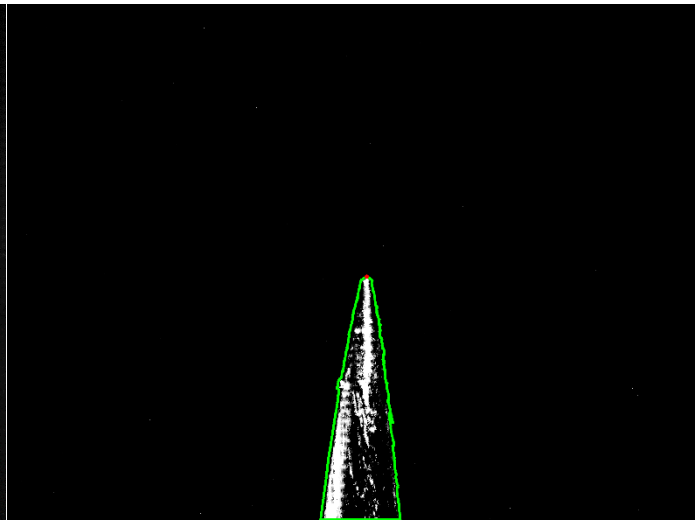

Figure 10: Image Results with contour and Top Pin Extrema (1723, 1306)

One of the major problems encountered along the beamlines is the centering and alignment of crystals and pins in the hutches. Currently, beamline scientists have to manually calibrate the motors to align a pin properly using cameras to provide visual confirmation of the sample. This results in a tedious and costly procedure that can be easily automated with the use of computer vision and robotics.

For this program, images of the pin are captured from the On-Axis Camera located in D Hutch at the IXS Beamline. The camera collects images of the pin, and motors help calibrate the position of the pin. The camera provides visual confirmation of the pin. The purpose of this particular application is to measure the position of both the x and y pixel position of the tip of the pin. By using this, we can measure the rotational difference of the pin. The application should discover and plot the sinusoidal regression that corresponds to the pixel positions of the pin tip. The program should use the results of the sine curve, specifically the amplitude and phase, to compute the correct adjustments for the motors to properly align the pin. A point and click GUI was made in order to facilitate the quick and easy calibration of the motors without manually adjusting it within the hutch or traversing different screens to adjust motor positions.

The program uses a flood fill to black out the entire background of the image. The image is then adjusted by a bitwise not operator, which negates the values of the pixels, i.e. white to black and vice versa. By contouring the pin, the program can discover the tip easily. With this point recorded, the program can let the pin rotate and construct a sine curve based upon the data. From this, the program can adjust motor positions to calibrate the pin so that it does not wobble when the motors are rotating.

For the point and click GUI, Images are fetched directly from EPICS, with the center of the image and the current selected position marked by two distinct cursors. A simple click will

adjust the x and y pixel positons of the cursor, and EPICS will update with the new values, using the stored motor counts to compute the necessary distance the pin must travel to reach the new point. This application will allow scientists to easily calibrate the pin within the camera frame without traversing multiple screens.

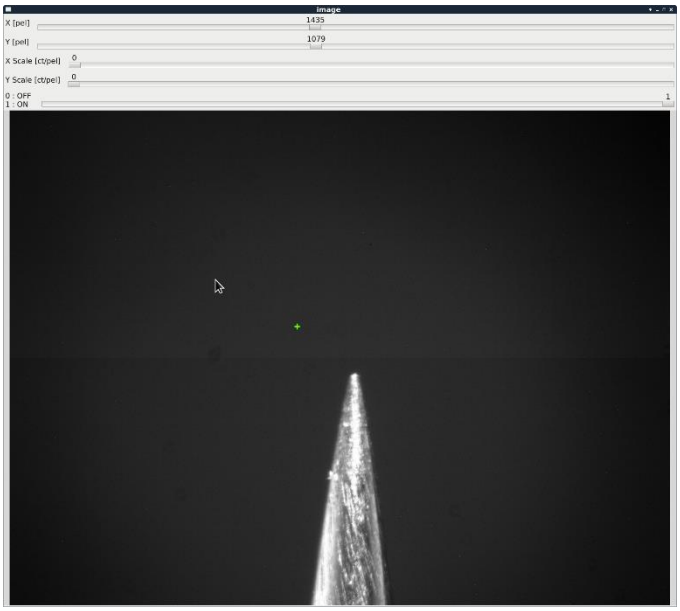

Figure 11: Screenshot of the point and click GUI, with track bars, live image, and green cross marking new position

D. ABBIX Beamlines: Pins in a Robot Gripper (AMX)

Computer vision can help assist users and scientists to the improper mounting of pins as they are picked up from a robotic gripper. This application was made to provide a detailed

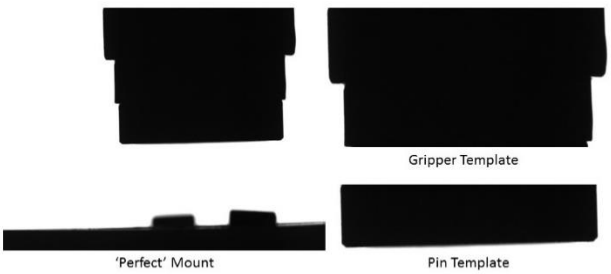

Figure 12: 'Perfect' template images

Figure 13: Case Samples for Pin and Gripper

analysis of a robotic gripper picking up a pin and to report to the user if an anomaly has

occurred, in order to prevent possible damage to the equipment or sample. The images for this application were provided by Jean Jakoncic, a beamline scientist for the Highly Automated Macromolecular Crystallography Beamline (AMX).

The purpose of this application is to find the pin and gripper region of interests (ROI), compare the apparatus to a set of 'perfect' templates, and to find kinks in order to check if the pin is properly mounted. The centroid is also discovered, along with the extreme points to provide assistance for alignment.

By thresholding the image and discovering the contours, the program can parse through the list of points and use logic checks to discover kink points along the contour. The points can be used to check for alignment if the pin and gripper are not entirely in the frame. Finally, template matching can quickly locate a bounding rectangle for the gripper and pin. The blue points are kink points, the red points are extrema, and the cyan point is the centroid. The red lines are vertical tangents, and the green line outlines the pin. The program stabilizes around 0.227 seconds per image.

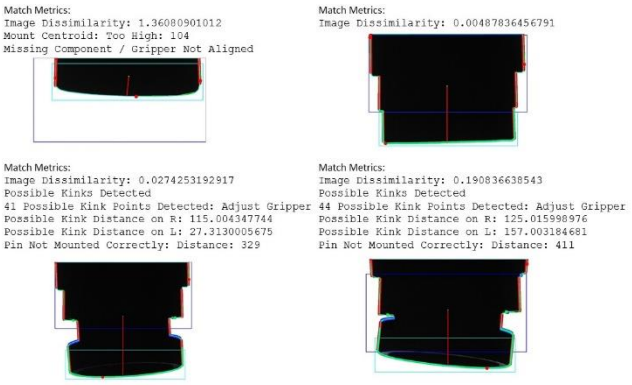

Figure 14: Image Results for the Pin/Gripper Analysis

Number of Images	Total Time	Average Time
1	0.544	0.544
2	0.751	0.3755
3	0.945	0.315
4	1.162	0.2905
5	1.361	0.2722
6	1.565	0.26083333
7	1.802	0.25742857
8	1.999	0.249875
9	2.192	0.24355556
10	2.429	0.2429
11	2.621	0.23827273
12	2.803	0.23358333
13	3.064	0.23569231
14	3.264	0.23314286
15	3.482	0.23213333
16	3.637	0.2273125
17	3.824	0.22494118
18	4.1	0.22777778
19	4.302	0.22642105

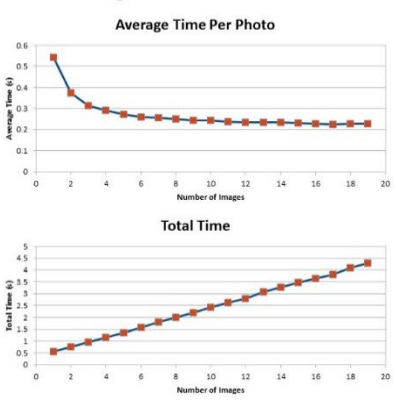

Figure 15: Time Charts for Pin/Gripper

E. ABBIX Beamlines: Crystal Goniostat Rotational Alignment (AMX)

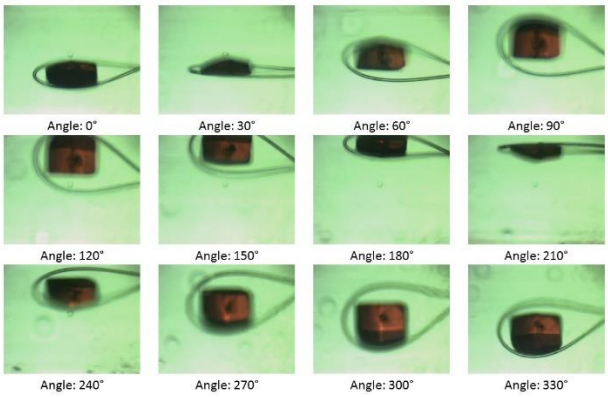

Figure 16: Crystal Images Every 30 Degrees

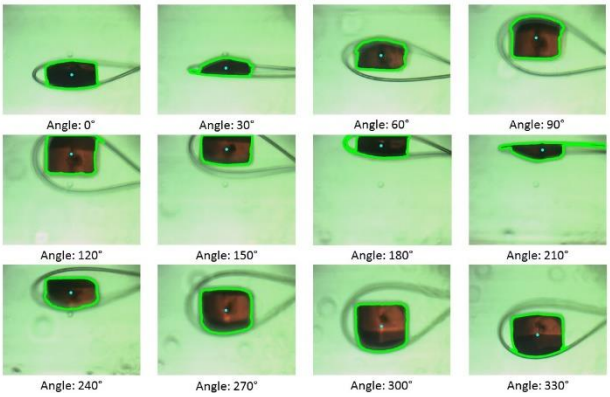

Figure 17: Results of Contouring, with Centroid Drawn

Crystal alignment in a goniostat is a time consuming process that can be handled by computer vision. At the AMX beamline, a crystal placed inside a loop rotates in a goniostat, with a full range of 360 degrees. The crystal is not centered so the crystal appears to wobble. The program processes images every 15 degrees for improved accuracy. By using an in range

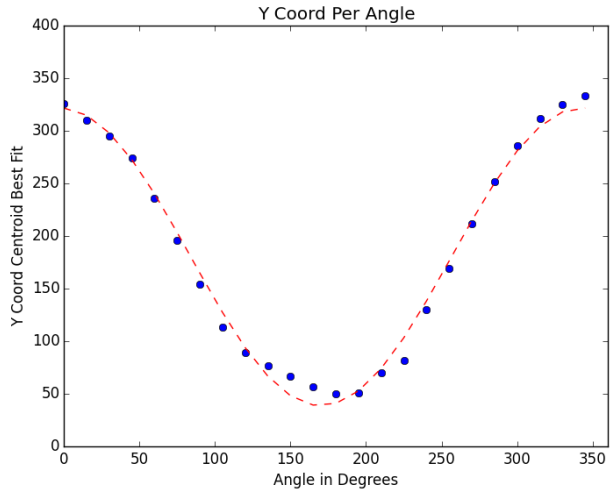

Figure 18:

Left: Graph Generated

Equation:

$$141.58 \times sin(angle + 1.61) + 180.19$$

Adjustment:

$$x = -\frac{MC}{PEL} \times Amplitude \times sin(phase)$$

$$y = -\frac{MC}{PEL} \times Amplitude \times cos(phase)$$

threshold, the only part of the image left is the crystal. With the background and extra noise eliminated by the thresholding, a contour can easily be found. From this contour, a centroid can be plotted.

The contour and centroid for the crystal is discovered. By plotting the y coordinate of the centroid against the angle, a sine curve is generated. By using the least squares optimization

procedure, the program finds the best fit sine curve to the given data. From the amplitude and phase of the best fit curve, the program can compute and send to EPICS the pin motor adjustments, allowing the motors to calibrate the crystal. The equation generated finds the best possible solution, with an amplitude and phase shift. Using certain formulas, the program can decompose the equation into the correct adjustments. MC stands for motor counts, and PEL stands for picture elements. MC/PEL is the conversion factor to convert pixel positions to actual measurements used for calibration. The total time taken for all 24 images was 0.858 seconds, with an average time per image of approximately 0.035 seconds. This will be useful in the quick alignment of crystals and can be applied to all beamlines.

IV. Conclusions

By building the infrastructure and general purpose wrapper libraries, beamlines can take full advantage of computationally efficient computer vision to help automate tedious processes along the beamlines. Computer vision can center samples in goniostats quickly. Programs can automate robotic movements and sample detection. Scientists can build applications to help with the assistance of alignment and calibration of samples, crystals, and beams along the beamlines. Users can discover and record data related to objects within an image. Programs can be developed to help prevent potential problems by alerting users of anomalies and mitigating damage to samples. With the integration of EPICS, and the newest version of OpenCV, this project integrates easily to the current systems installed on the beamlines and can efficiently fetch and interpret image data directly from the cameras along the beamline.

V. Acknowledgements

This project was made possible by the support from my mentor Kazimierz Gofron. Images from the IXS Beamline are courtesy of Yong Cai, Alessandro Cunsolo, and Alexey Suvorov. Images from the ABBIX Beamline AMX are courtesy of Jean Jakoncic. This project was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internships Program (SULI).

VI. References

[1] (Intel), Kay M. "Intel® Integrated Performance Primitives (Intel® IPP)." *Intel® Software*. Intel, 2016. Web. 09 Aug. 2016. <https://software.intel.com/en-us/intel-ipp>.

[2] Intel. "Threading Building Blocks." *Threading Building Blocks*. Intel Corporation, 2016. Web. 09 Aug. 2016. <https://www.threadingbuildingblocks.org/>.

[3] Johnson, Andrew N. "EPICS - Experimental Physics and Industrial Control System." *Argonne National Laboratory*. Argonne National Laboratory, 31 May 2016. Web. 9 Aug. 2016. <www.aps.anl.gov/epics/>.

[4] OpenCV Developer Team. "About OpenCV." *OpenCV*. Itseez, 2016. Web. 9 Aug. 2016. <http://opencv.org/about.html>.

[5] Yong Cai, Alessandro Cunsolo, Alexey Survov. "IXS Schematic Diagram." *Inelastic X-Ray Scattering Beamline (10-ID)*. National Synchrotron Light Source II, Brookhaven National Laboratory.