



BERKELEY LAB

Bringing Science Solutions to the World



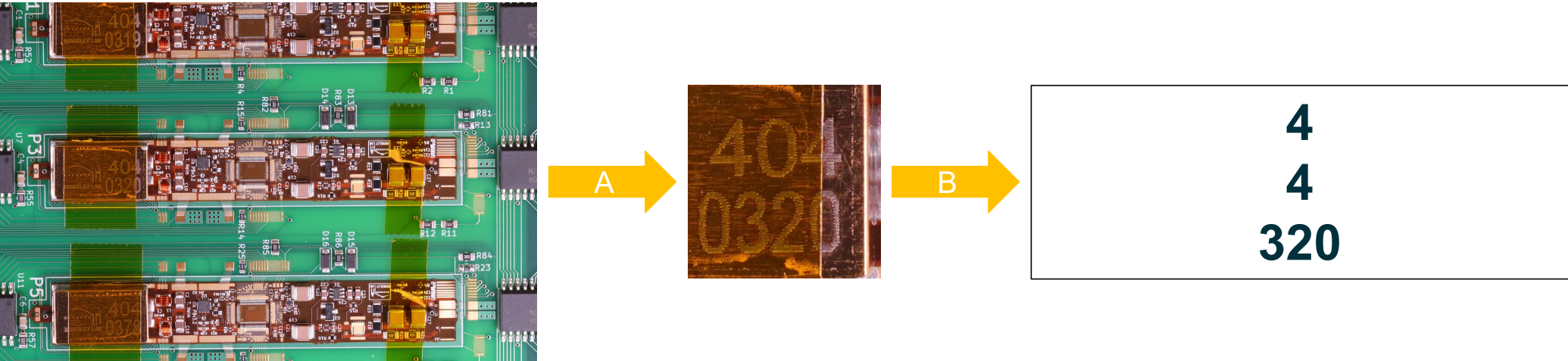
U.S. DEPARTMENT OF
ENERGY
Office of Science

Shieldbox Number ML

Samantha Kelly

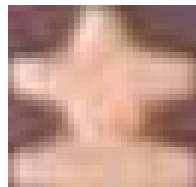
(under direction of Timon Heim and Zhicai Zhang)

Objective 1: Shieldbox



1a: Cropping the shieldbox

- Input image taken during the visual inspection stage of the [Powerboard Quality Control Procedure](#)
- Goal of algorithm is to identify shieldbox top-left and bottom-right corners using fiducials
 - Crops image using location of corners
- Good crop should eliminate all other text and clearly include all 7 shieldbox numbers



One of the five fiducials on a powerboard

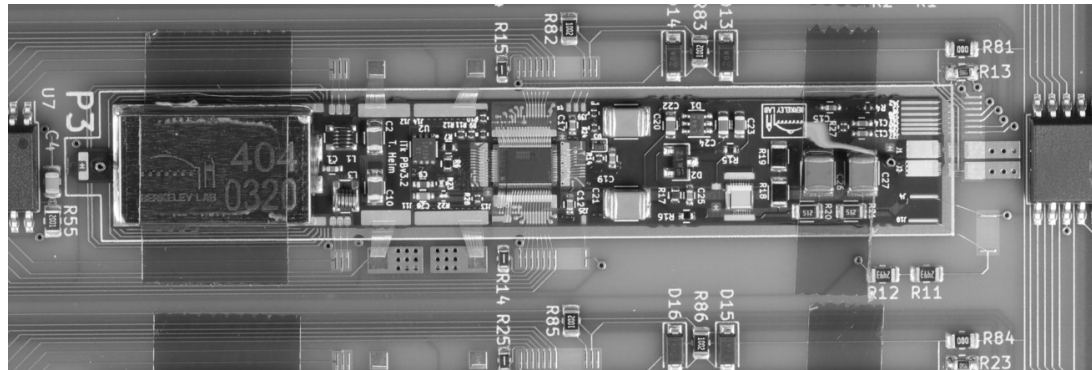


A discolored fiducial

1a: Cropping the shieldbox

Opening the input image and template

- Open the image using [`cv2.imread`](#)
- Make the image black and white using [`cv2.IMREAD_GRAYSCALE`](#)
- Crop the image to the pixel range (1000:3000, 0:6000)
- Open the template using [`cv2.imread`](#)
- Make the template black and white using [`cv2.IMREAD_GRAYSCALE`](#)



1a: Cropping the shieldbox

Locating fiducials

- Using `eval('cv2.TM_CCOR_NORMED')` as method for template matching
- Check how close fiducial template is to actual fiducials using [cv2.matchTemplate](#)
- Record pixel coordinates of regions matching the template
 - Arbitrary threshold of “matching” set high enough to remove false fiducials, but low enough to account for variation

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

with mask:

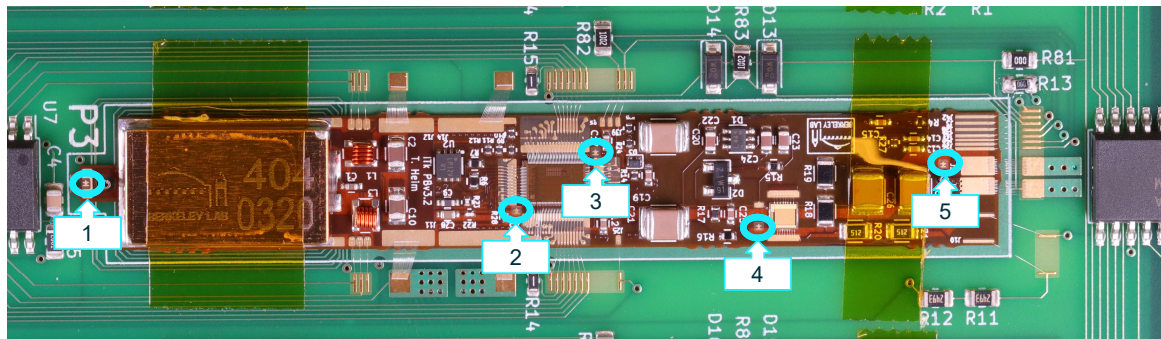
$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') \cdot I(x + x', y + y') \cdot M(x', y')^2)}{\sqrt{\sum_{x', y'} (T(x', y') \cdot M(x', y'))^2 \cdot \sum_{x', y'} (I(x + x', y + y') \cdot M(x', y'))^2}}$$

Template matching operation for TM_CCOR_NORMED

1a: Cropping the shieldbox

Determine pair of fiducial

- Use the coordinates of the first two fiducials stored
- Find the distance between the x- and y-coordinates of the fiducials respectively
- Find the ratio between the x- and y-coordinates of each fiducial, and the y-coordinates of both fiducials
- Using these values, establish parameters to determine which fiducials are stored
 - 10 cases (1-2, 1-3, 1-4, 1-5, 2-3, 2-4, 2-5, 3-4, 3-5, 4-5)



1a: Cropping the shieldbox

Sort and remove duplicate fiducials

```
70 priorpt = (0,0) #initiate as upperleft corner of image
71 for pt in zip(*loc[::-1]): #for every located fiducial
72     if 600 < pt[0] < 2000: #if it's in the range of the shieldbox (where there should be no fiducials),
73         continue #skip
74     elif pt[0] > 5500: #if it's way to the right of the far-right fiducial,
75         continue #skip
76     elif abs(priorpt[0] - pt[0]) > 100: #if the located fiducial is far enough away from the prior pt to not be the same one
77         overlap = 0 #initiate variable
78         for i in range(len(appxloc)): #for all potential fiducials
79             if abs(pt[0] - appxloc[i][0]) < 50: #if the current pt and the iterated pt in the list are close enough
80                 overlap += 1 #increase variable
81             if overlap == 0: #if the current pt doesn't already exists in the list i.e. is far enough from preexisting pts
82                 appxloc.append(pt) #add it to the list
83         priorpt = pt #update the current pt to prior pt
84 appxloc.sort(key=lambda a: a[0]) #sort the list by x coordinates
```

- If less than five fiducials found, repeat Slides 5-7 using discolored fiducial as template image

Attempt 1: Intuition

Crop shieldbox

- Use determined fiducial pair and coordinate values/distances/ratios to find top-left and bottom-right corners
- Corner = (fiducial_x + constant*dx, fiducial_y + constant*dy)
 - Constants determined experimentally
- Crop shieldbox from corner to corner
- Full code available on [GitLab](#)
- Benefits:
 - It kind of works (crops shieldbox region for 238/239 images)
 - What I initially thought of
- Why this failed:
 - Not consistent enough (i.e. varying crop sizes)
 - Doesn't account for rotation of image or zoom correctly



vs.



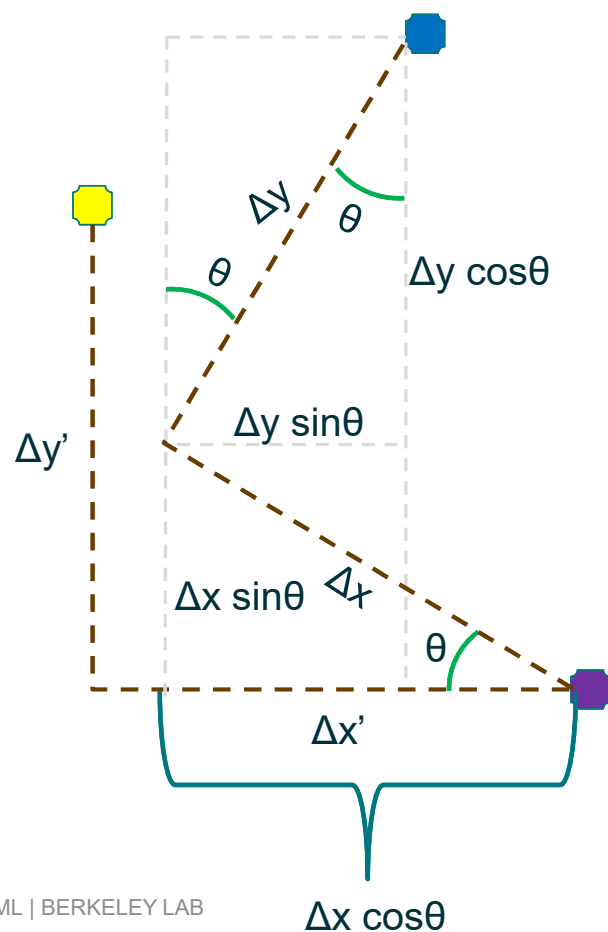
Rotating the image

Let's make a function

```
11 def rotate_image(image, angle): #I wonder what this does
12     height, width = image.shape[:2] #get image dimensions
13
14     rotation_matrix = cv.getRotationMatrix2D((width / 2, height / 2), angle, 1) #calculate the rotation matrix
15
16     rotated_image = cv.warpAffine(image, rotation_matrix, (width, height)) #apply the rotation to the image
17
18     return rotated_image #I wonder what this is
```

- Need to get angle

Angle Attempt 1: Trig



“Solution”:

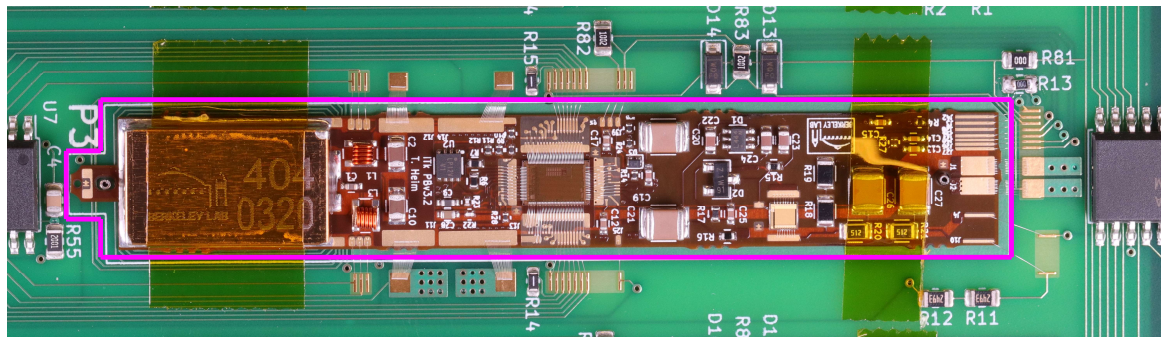
- $\Delta x' = \Delta x \cos \theta - \Delta y \sin \theta$
- $\Delta y' = \Delta x \sin \theta + \Delta y \cos \theta$
 - Sub trig for 1st order Taylor

Why this failed:

- It was wrong
- That is not how geometry works
- Everything but the solution eqs was fine
 - But even the diagram assumes the image rotates around the second fiducial (false)

Angle Attempt 2: Finding the white border

- Use [cv2.HoughLinesP](#) to locate longest white line on image
- Find angle between that line and the horizontal
- Why this failed:
 - Not consistently locating the white border
 - Too many other lines on image
 - Reducing the pixel range scanned did not help



Angle Attempt 3: Revenge of the Trig

Vector time

- When making slides for this presentation, I realized we were doing the trig incorrectly

🤖 Asked a [robot](#) for help

```
178 dot_product = dx_original * dx_rotated_scaled + dy_original * dy_rotated_scaled
179 cross_product = dx_original * dy_rotated_scaled - dy_original * dx_rotated_scaled
180 rotation_angle = np.arctan2(cross_product, dot_product)*180/np.pi
181
182 rotated_image = rotate_image(img, rotation_angle)
```

- Why does this work?
 - Can think of our two sets of ordered pairs as vectors
 - $\tan\theta = \frac{\|v \times w\|}{v \cdot w}$
 - [np.arctan2](#) takes the arctangent of a value with a numerator and denominator

Correcting the zoom

It'd be too easy if the camera zoom was the same every time

```
244     scale_factor = np.sqrt(dx_original**2 + dy_original**2) / np.sqrt(dx_rotated**2 + dy_rotated**2)
245
246     new_width = int(image_color.shape[1] * scale_factor) #use the scale to scale
247     new_height = int(image_color.shape[0] * scale_factor)
248
249     resized_image = cv.resize(rotated_image, (new_width, new_height))
```

- Why does this work?
 - Scale factor is the ratio of the distances between the original image points and the crooked/zoomed image points

The end...?

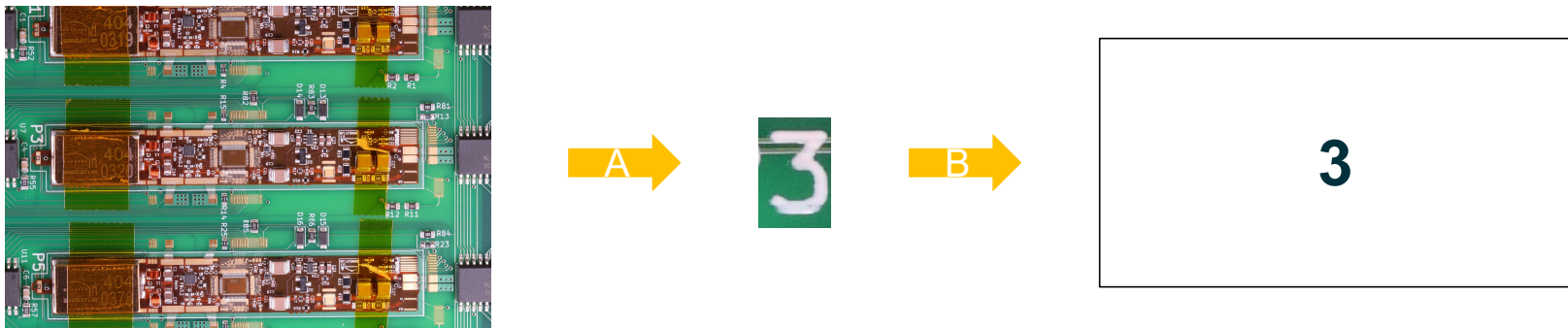
At least for Sam at LBNL

- Surprise! I'm leaving 🤪
 - Starting grad school at Davis in the fall
 - My two-year reign of being funny at the lab concludes
- But this project isn't done?
 - You're right
 - Steps left to do:
 - Crop the zoomed and straightened image
 - Can either be done:
 - » by aligning the first registered fiducial with the first registered fiducial on a perfectly-straight image
 - » by manually assessing the cropped region based on how far the fiducial pairs are from each other (see Slide 8)

The end...?

At least for Sam at LBNL

- More steps left to do:
 - Run a machine learning algorithm to identify the shieldbox numbers
 - A skeleton code will be uploaded to [Github](#) by EoD
 - » Does not currently run due to lack of cropped images thus far, but has general idea
 - Transfer these steps to Objective 2: Powerboard #
 - Should be extremely easy, just a simpler version of cropping the shieldbox numbers and a simpler ML algorithm (since the number is printed vs. etched on)



Questions now?

For questions later, message on: Mattermost (@sakelly), email (samanthakelly@berkeley.edu)

