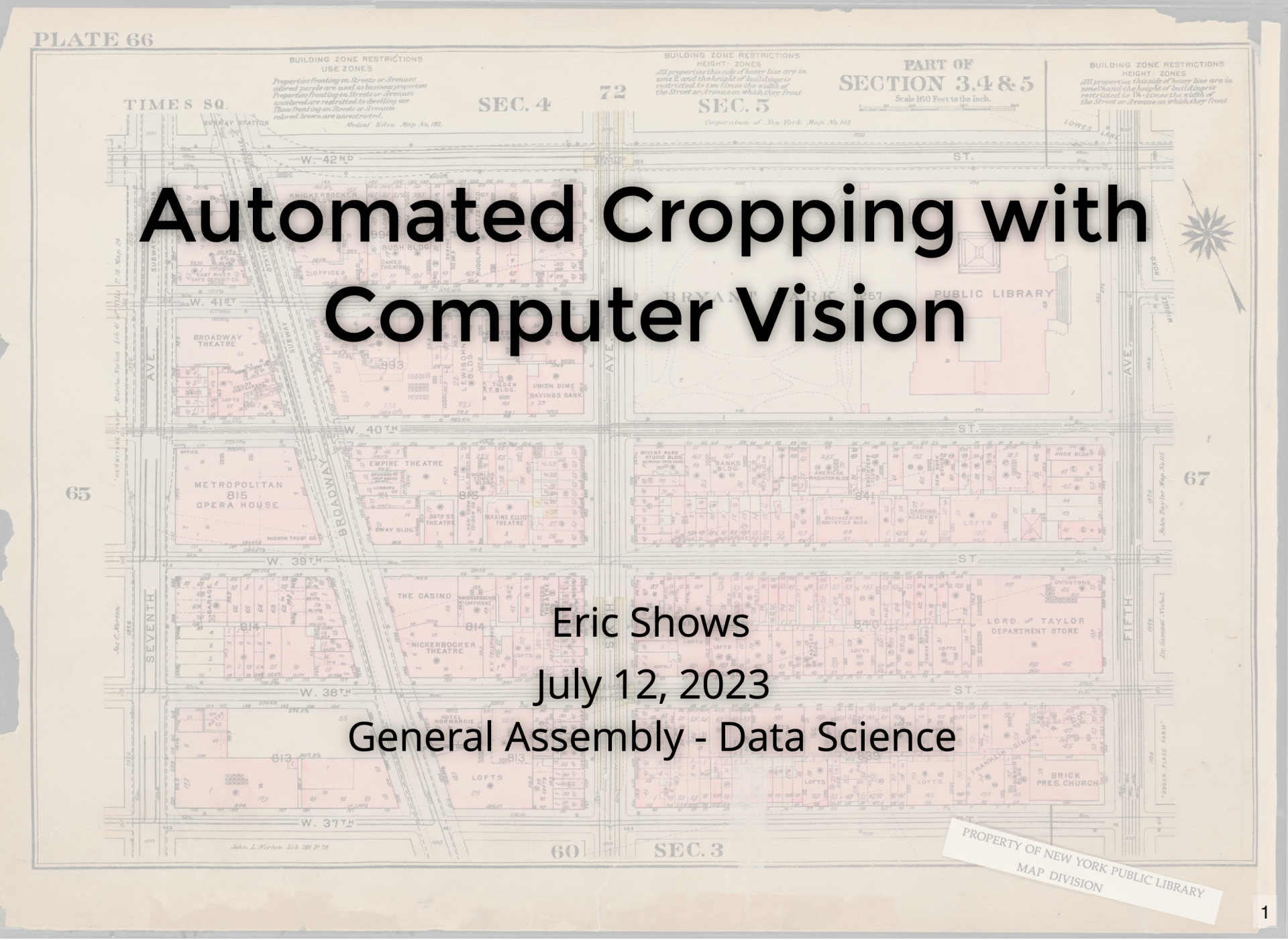


Automated Cropping with Computer Vision

Eric Shows

July 12, 2023

General Assembly - Data Science

Problem Statement

There are 1,037,563 captures (32TB) in NYPL's digital repository that are missing a web-ready high-resolution derivative. In order to create these missing images, computer vision techniques were used to compare a lower resolution cropped image with the uncropped master tif file to create a new set of high-resolution images.

Data Gathering

Select Sample Data Set

Maps of North America are currently being served by IIIF image server and preexisting georeference data provides an opportunity to connect to third party sites like allmaps.org

Sample image set - 373 matched pairs of low-res .jpg and high-res .tif files

Download Images

Query SQL database for qualifying sets of uncropped tif files and low-res cropped files where a high-res cropped file is missing

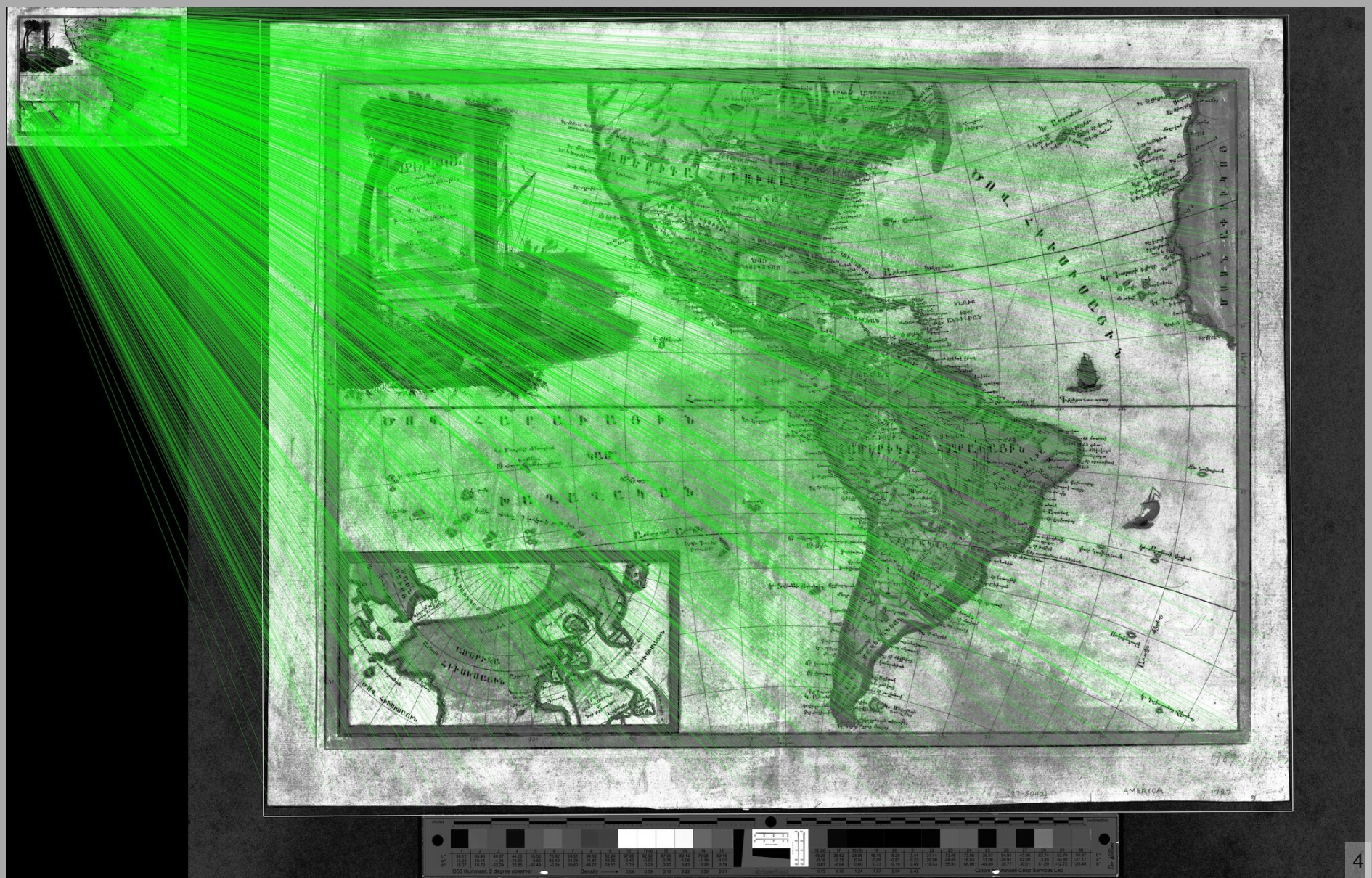
```
SELECT FILE_ID
FROM file_store  AS u
LEFT JOIN file_store AS s
ON u.FILE_ID = s.FILE_ID AND s.TYPE = 'j'
WHERE u.TYPE = 'u' AND s.TYPE IS NULL;
```

Detect Features

Use SIFT feature detection and FLANN matching algorithms to find key points of comparison between images

Detect boundary of target file (low-res .jpg) in source image (high-res .tif)

Observing the Extracted Features



Extract and Observe Features

Extract Features with SIFT and FLANN

- Matched features are detected using an approximate nearest neighbors algorithm

- Matched features are stored in a list once declared good matches

```python
# Initiate SIFT detector
sift = cv2.SIFT_create()

# Find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(temp_img_eq, None)
kp2, des2 = sift.detectAndCompute(map_img_eq, None)

FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=50)

flann = cv2.FlannBasedMatcher(index_params, search_params)

# Find matches by knn which calculates point distance in 128 dim
matches = flann.knnMatch(des1, des2, k=2)

# Store all the good matches as per Lowe's ratio test
good = []
for m, n in matches:
    if m.distance < 0.7 * n.distance:
        good.append(m)
```

```python
# find homography
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
matchesMask = mask.ravel().tolist()

h, w = temp_img.shape
pts = np.float32([[0, 0], [0, h-1], [w-1, h-1],
                  [w-1, 0]]).reshape(-1, 1, 2)
dst = cv2.perspectiveTransform(pts, M)  # matched coordinates

map_img = cv2.polylines(
    map_img, [np.int32(dst)], True, 255, 3, cv2.LINE_AA)
```

Use Detected Features to Draw Lines

cv2.polylines draws lines between each set of matched features

Make Observations

Matches are plentiful and indicate robust comparison capabilities

Observe Target Boundary Scaled to Source Image

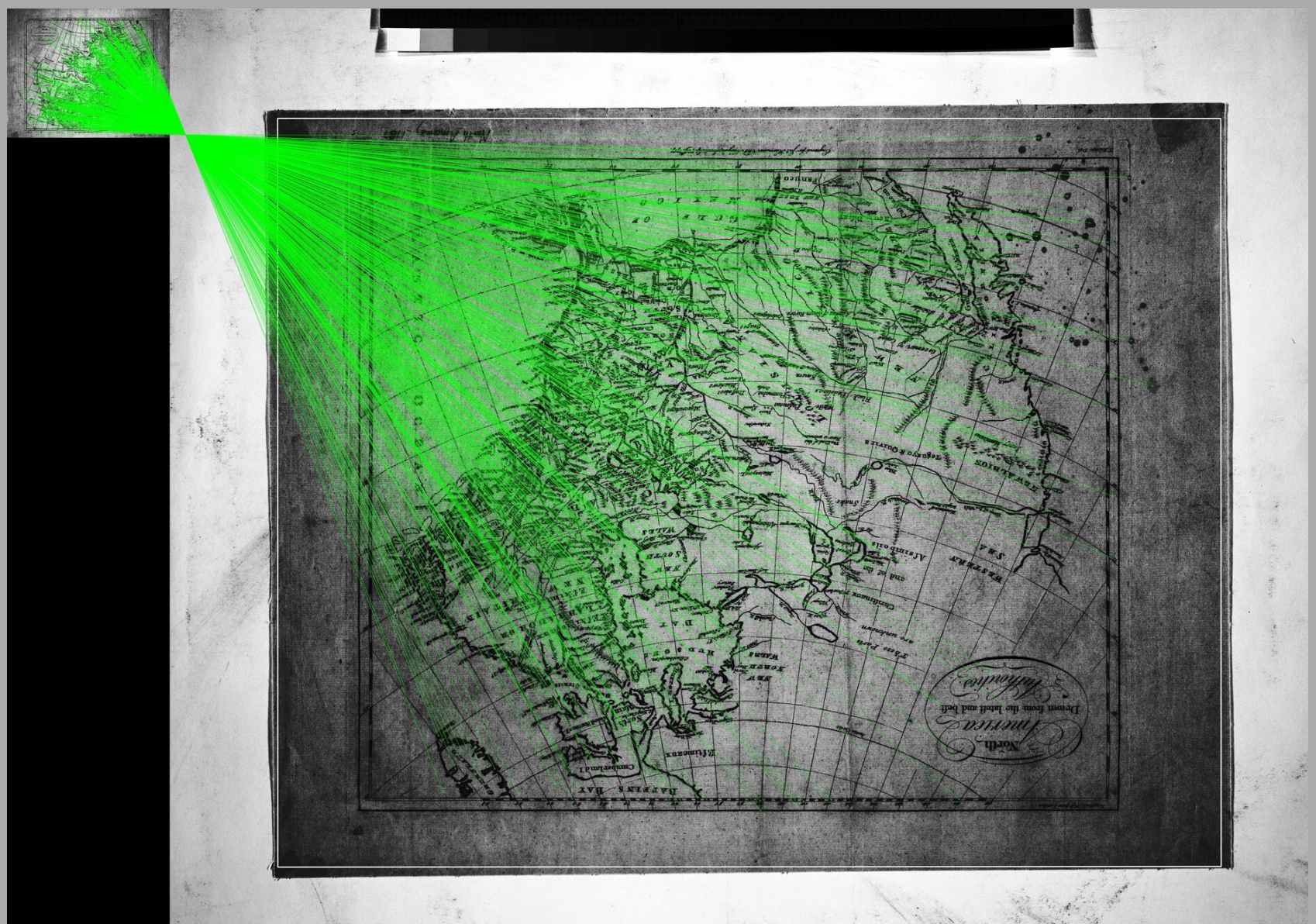
Use Detected Features to Draw Target Boundary on Source Image

Corners of target are easily detected by SIFT and FLANN algorithms

Make Observations

Visual inspection of modified data set copy indicates SIFT and FLANN are viable techniques for recognizing a target boundary that can be used to create new high-res cropped images

Compare Target and Source Orientation



Compare Target and Source Orientation

Image Orientation Inconsistencies

Source and target images are often not oriented in the same direction

Use Detected Features from SIFT and FLANN for Comparison

Build a comparison tool to examine features and detect orientation

Perform Rotation at 90 Degree Intervals

Reorienting images to ensure source and target images match amounts to a preprocessing step, so rotation doesn't need to be precise

Rotating at 90 degree intervals allows for easier image comparison in preview applications

Observing Rotation Angle

Observing Rotation Angle

Initial Tests Yield Incorrect Rotation in Certain Cases

Most maps are already rotated to match the image plane and don't require additional rotation

Some maps will not crop with correct rotation no matter what parameters are tuned or modified

Inspect Detected Corner Points

To observe why this might be, draw red circles at detected corner points and compare with green bounding boxes

which represent implied or predicted crop edges

Observe and Measure Difference

Red circles and green boxes align well for most images but not at all for others

Detected corner points drawn with red circles will need an angle calculation added to crop with correct rotation

Correcting Rotation Angle

Calculate Center of Region of Interest Before Rotation

In order to rotate images correctly during cropping process, rotation of crop corners must be calculated from

center point of the region of interest (green box)

```
# Rotate the transformed corners based on the estimated rotation angle
center_x = (transformed_corners[0, 0] + transformed_corners[2, 0]) / 2
center_y = (transformed_corners[0, 1] + transformed_corners[2, 1]) / 2
rotation_matrix = cv2.getRotationMatrix2D((center_x, center_y), rotation_deg, 1.0)
rotated_corners = cv2.transform(np.array([transformed_corners]), rotation_matrix).squeeze()

# Find the new minimum and maximum coordinates after rotation
x_coords = rotated_corners[:, 0]
y_coords = rotated_corners[:, 1]
x_min, x_max = np.min(x_coords), np.max(x_coords)
y_min, y_max = np.min(y_coords), np.max(y_coords)

# Round the coordinates to integers for cropping
x_min, y_min, x_max, y_max = int(x_min), int(y_min), int(x_max), int(y_max)

# Calculate the width and height of the output image based on the rotated region of interest
output_width = x_max - x_min
output_height = y_max - y_min
```

Handling Negative Values in Cropping Coordinates

Handling Negative Values in Cropping Coordinates

```
# Calculate the expansion amounts for the canvas
x_expansion = max(-np.min(transformed_corners[:, 0]), 0)
y_expansion = max(-np.min(transformed_corners[:, 1]), 0)

# Expand the canvas and fill with white color
expanded_width = int(rotated_image.shape[1] + x_expansion)
expanded_height = int(rotated_image.shape[0] + y_expansion)
expanded_image = np.ones((expanded_height, expanded_width, 3), dtype=np.uint8) * 255

# Calculate the offset for the original image on the expanded canvas
x_offset = x_expansion
y_offset = y_expansion
```

Expand Canvas and Scale Coordinates with Offset

Negative coordinates appear when a target image is visually bigger than it's scaled source (pixels were added during original cropping procedure)

Add white space to source image when a negative coordinate is detected to accommodate and offset negative values

Comparing RMSE Between Methods

Method

- Downscale final output cropped images to match long dimension in pixels of original low-res target .jpg files

- Use Root Mean Square Error to measure pixel by pixel differences between original cropped files and final output files

Automated Crop Detection without Rotation Correction

Average RMSE grayscale: 9.380575206422174

Average RMSE rgb color: 44.406036

Calculated Rotation Correction Method

Average RMSE grayscale: 8.404186285516756

Average RMSE rgb color: 17.577888

What's Next?

Optimize Pipeline for Speed

- FAST Algorithm for Corner Detection may offer some speed advantages

- Refactor application to only detect features once

```
Average Points Detected:
SIFT: 497.6751336898396
SURF: 497.6751336898396
ORB: 497.6751336898396
FAST: 175230.24598930482

Average Speed of Detection (seconds):
SIFT: 0.11343521644724881
SURF: 0.1137535814295478
ORB: 0.11442342767103471
FAST: 0.05400806314804975
```

Explore Rotation Techniques

- RMSE of results seems to indicate cv2.warpAffine may be introducing image distortion

Experiment with Clustering Techniques

- 1,037,563 eligible captures represent many different material types that may need different solutions for cropping