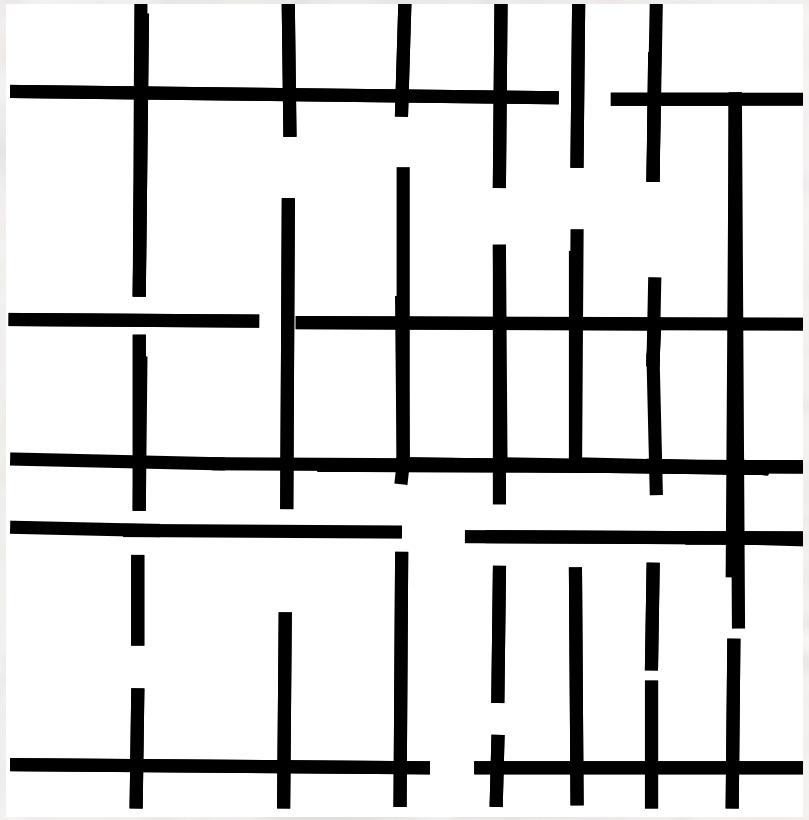
Converting past to present:

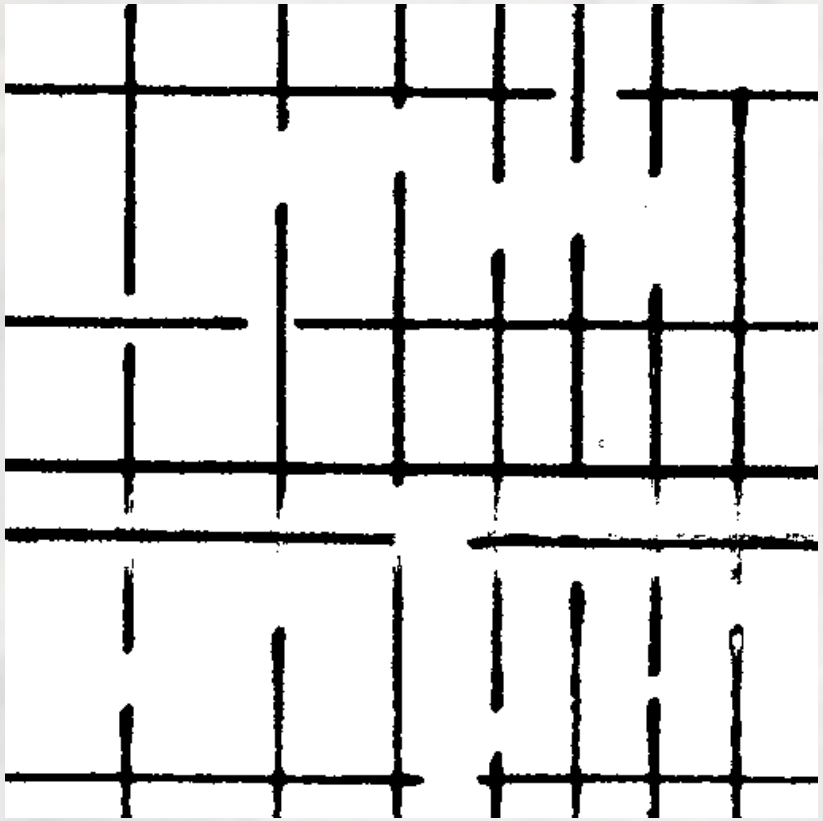
Turning historical hand-drawn architectural designs into vectors using autoencoder and corner detection algorithms

Mark Jeremy G. Narag
National Institute of Physics, University of the Philippines Diliman 1101
mnarag@nip.upd.edu.ph

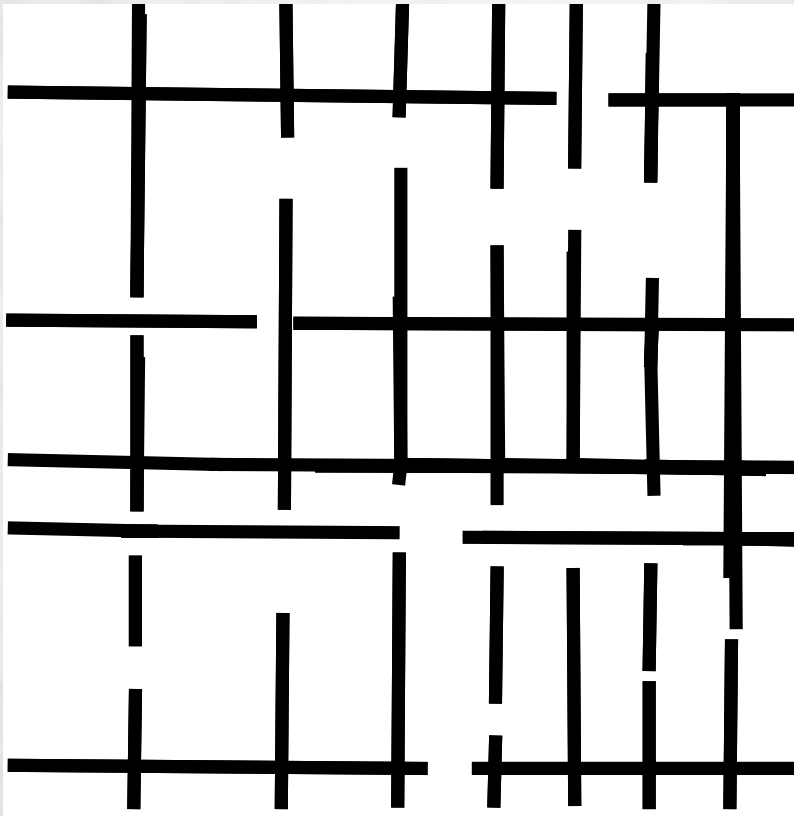
GOAL: Convert raster image to vector



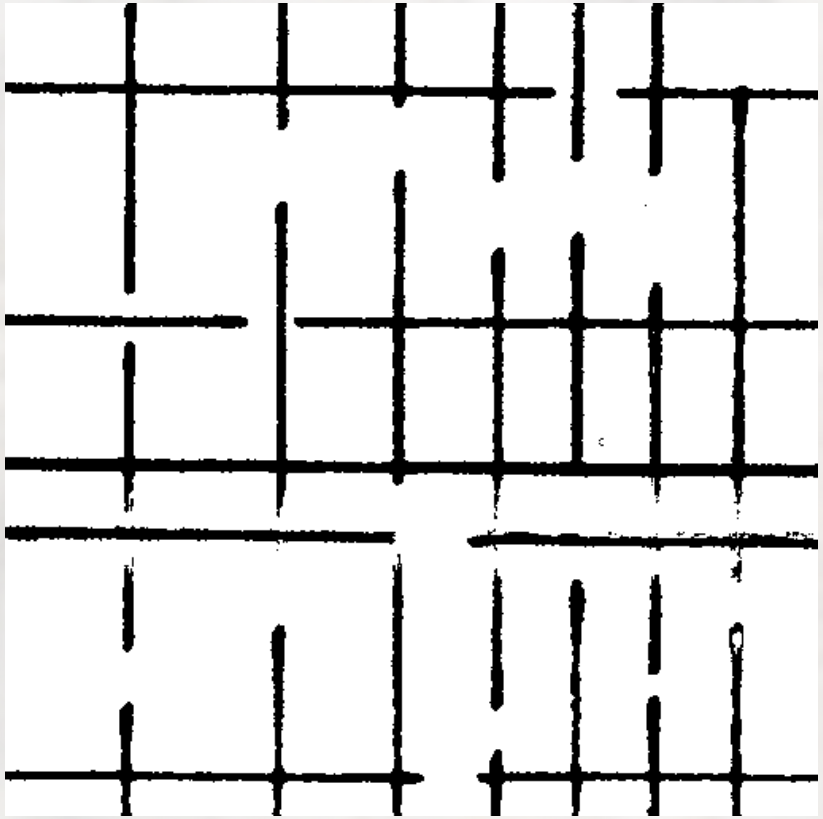
GOAL: Convert raster image to vector

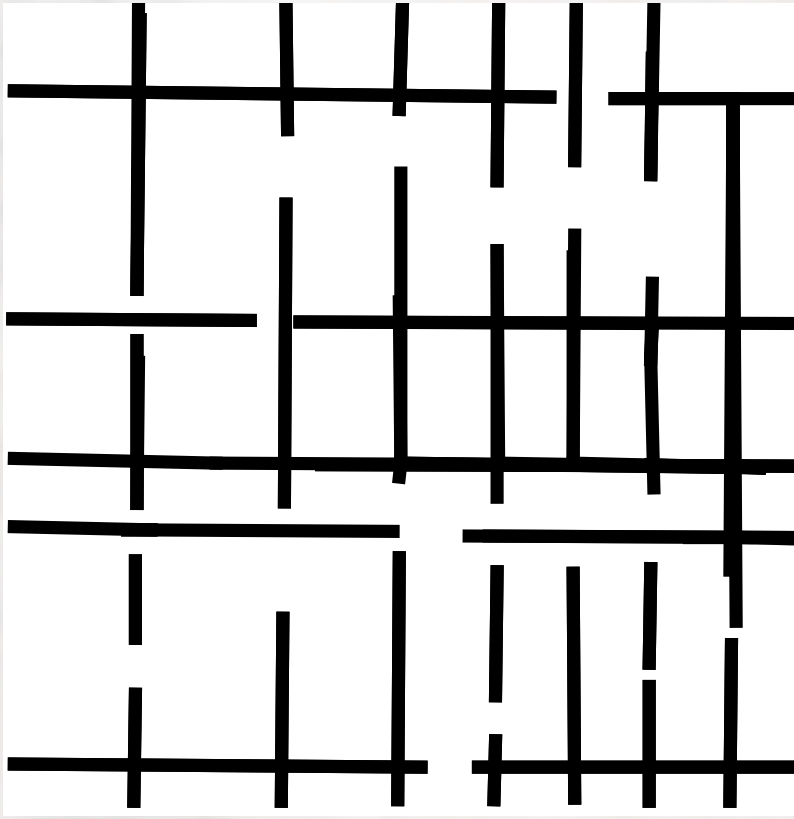


STEP 1: ”Cleaning” it through binarization



STEP 2: Vectorization through corner detection





NOVELTY:

- For binarization, first to attempt high resolution image.
- For data, first to attempt hand-drawn architectural designs.
- For task, first to attempt vectorization of severely stained hand-drawn architectural designs.

Agriculture and Commerce Building
Designed: 1938
Architect: Antonio Toledo
Constructed: 1940

Image from: National Museum of the Philippines, National Museum of Natural History, https://www.nationalmuseum.gov.ph/our-museums/national-museum-of-natural-history, accessed July 2023

Agriculture and Commerce Building
Designed: 1938
Architect: Antonio Toledo
Constructed: 1940

Image from: National Museum of the Philippines, National Museum of Natural History, https://www.nationalmuseum.gov.ph/our-museums/national-museum-of-natural-history, accessed July 2023

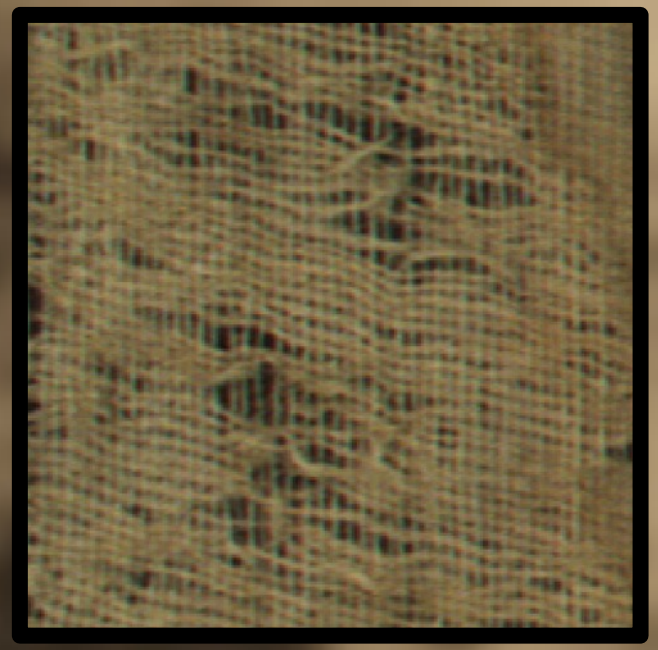
Natural Museum of Natural History (2016)

- planning to convert it started in 2012

Image from: *https://en.wikipedia.org/wiki/National_Museum_of_Natural_History_%28Manila%29#/media/File:National_Museum_of_National_History_(Manila)_in_2019.jpg*, accessed July 2023

Original 26 architectural drawings from DPWH

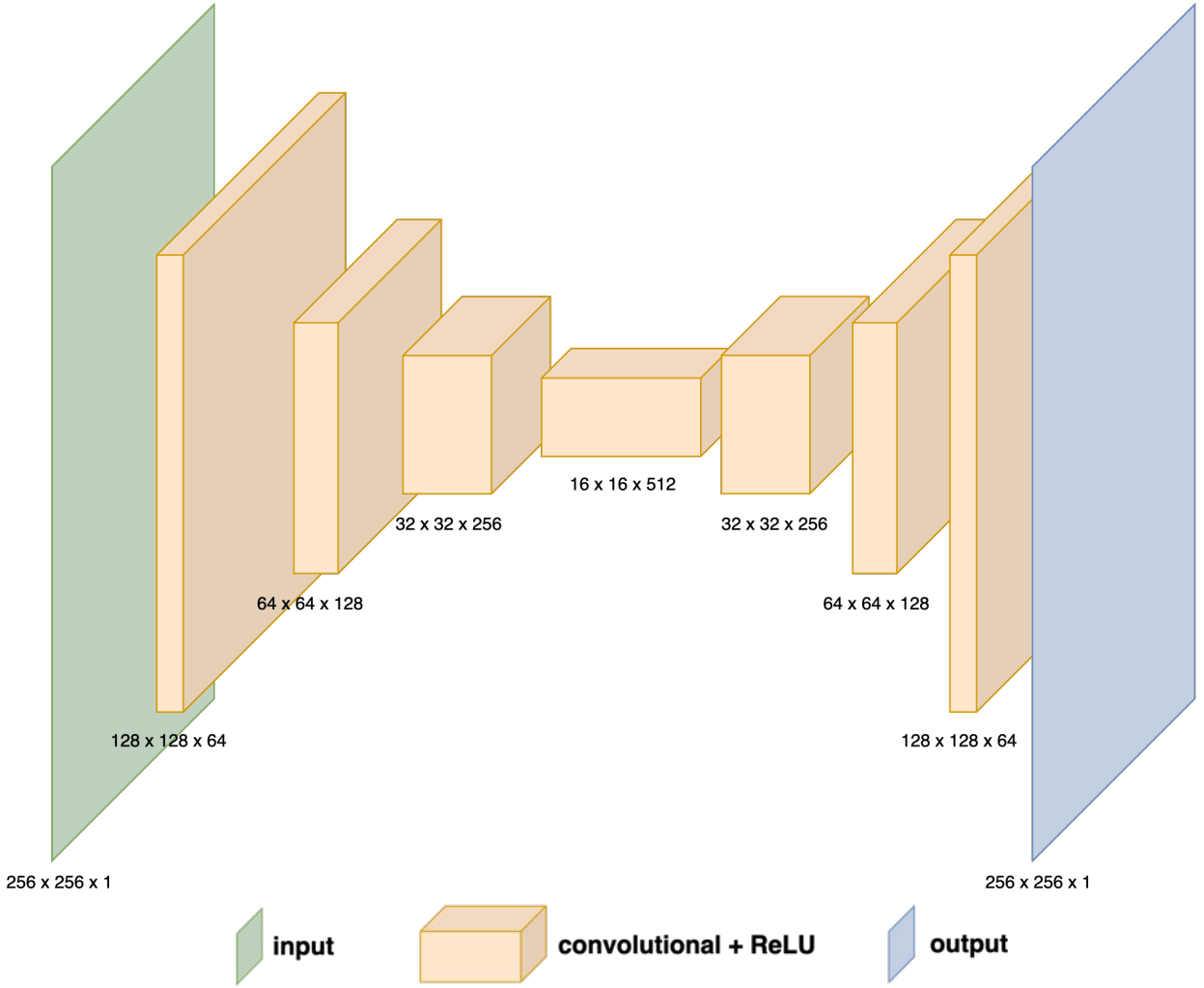
Our group digitally scanned them using NIJI (a digital scanner from Kyoto University)



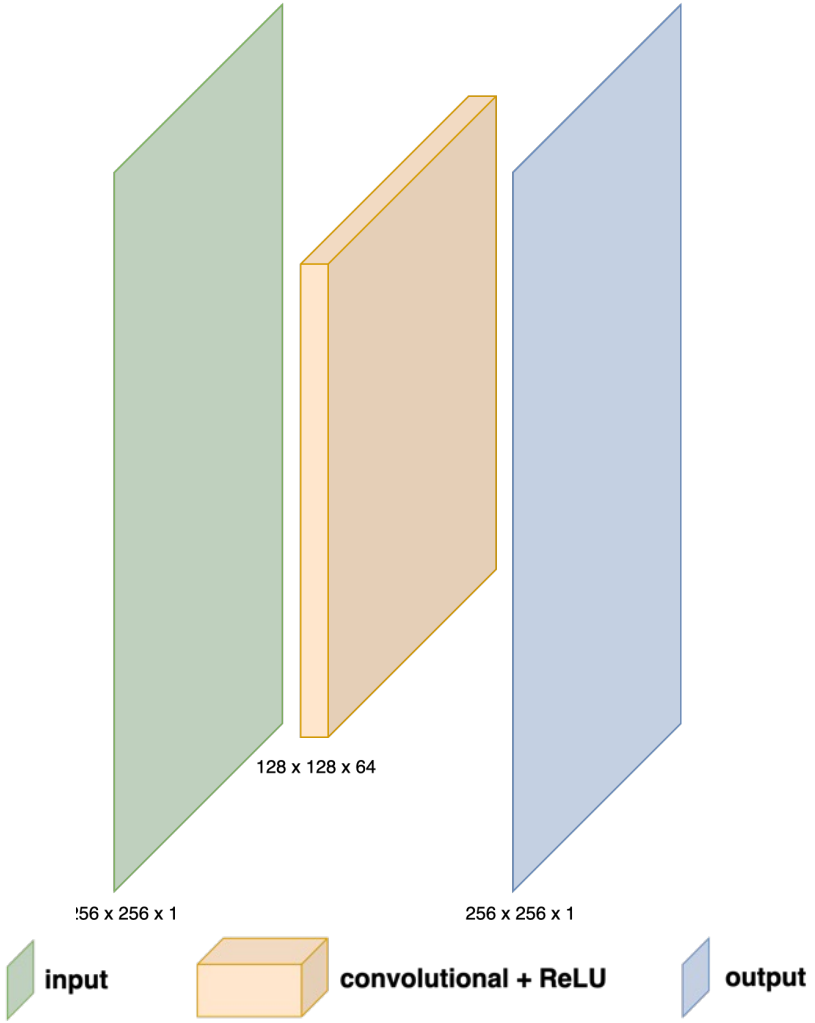
470 DPI

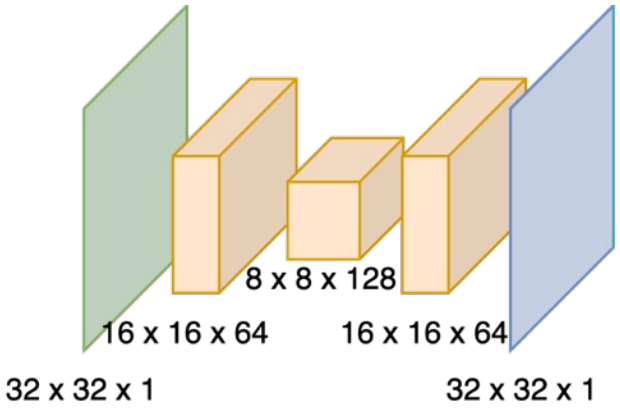
FIRST STEP IN VECTORIZATION IS “CLEANING” THEM

Revisited it after 10 years in the hopes of finding a universal solution using autoencoder

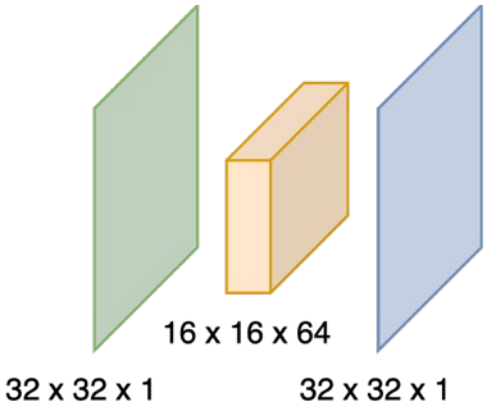


Revisited it after 10 years in the hopes of finding a universal solution using shallow autoencoder



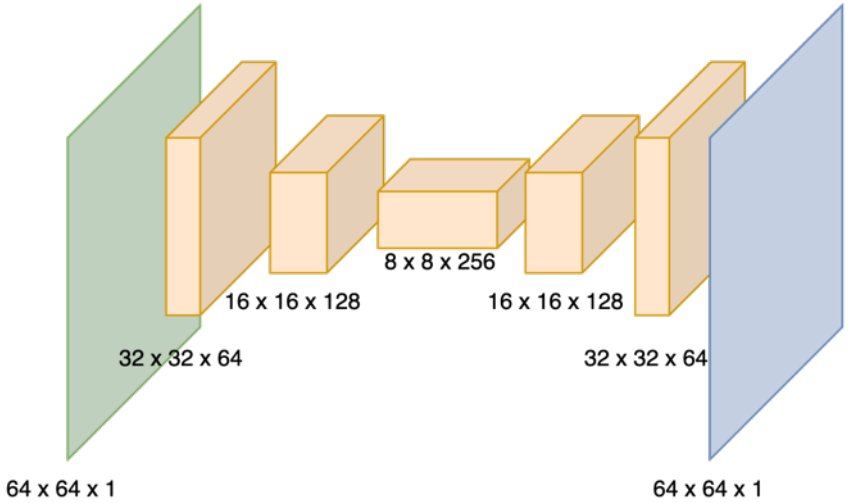


MODEL 1

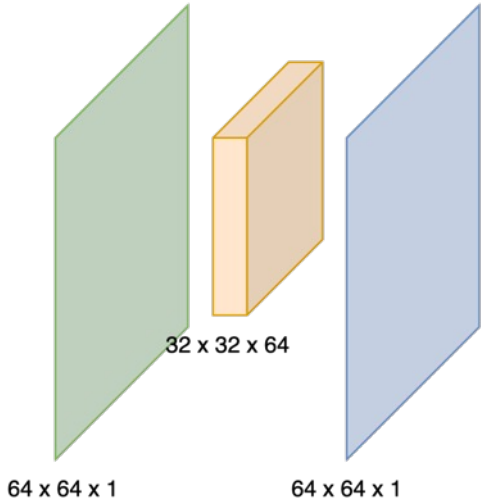


MODEL 2

kernel size 3 × 3
stride of 2
ReLU
adam solver
learning rate of 0.0001

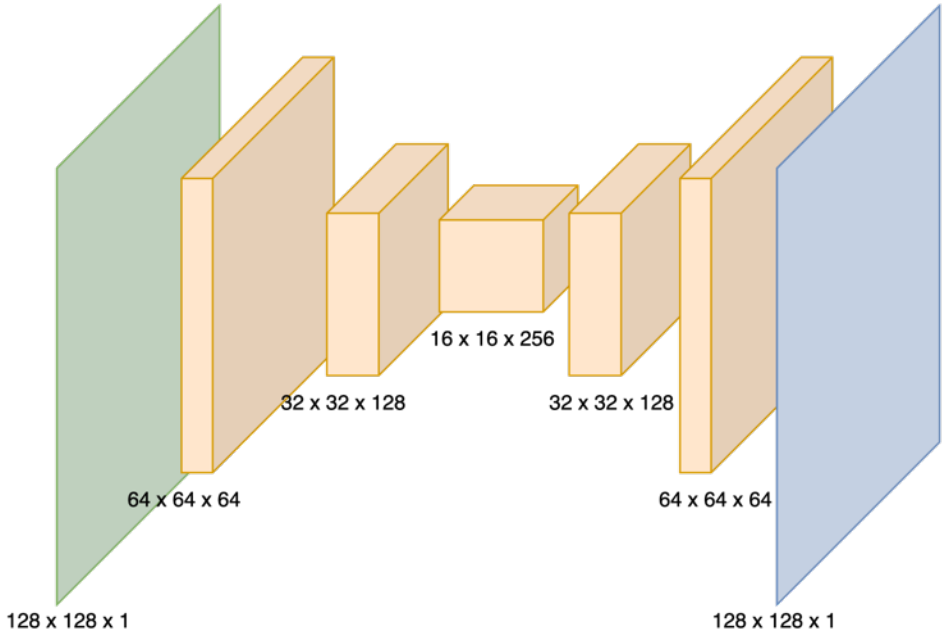


MODEL 3

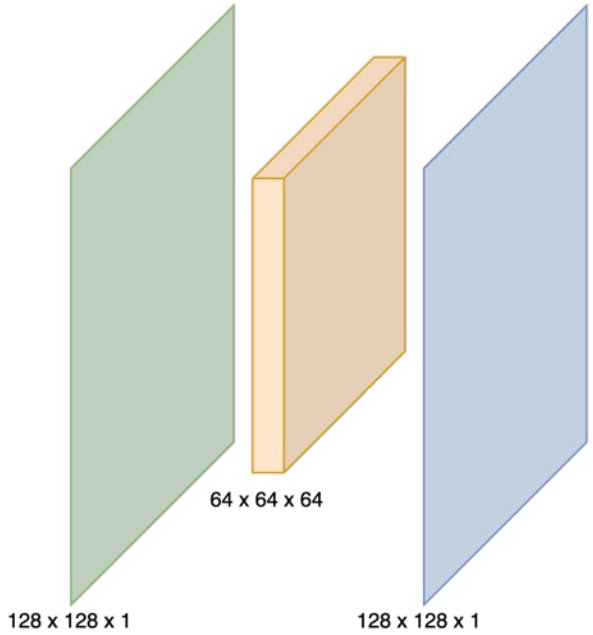


MODEL 4

kernel size 3 × 3
stride of 2
ReLU
adam solver
learning rate of 0.0001

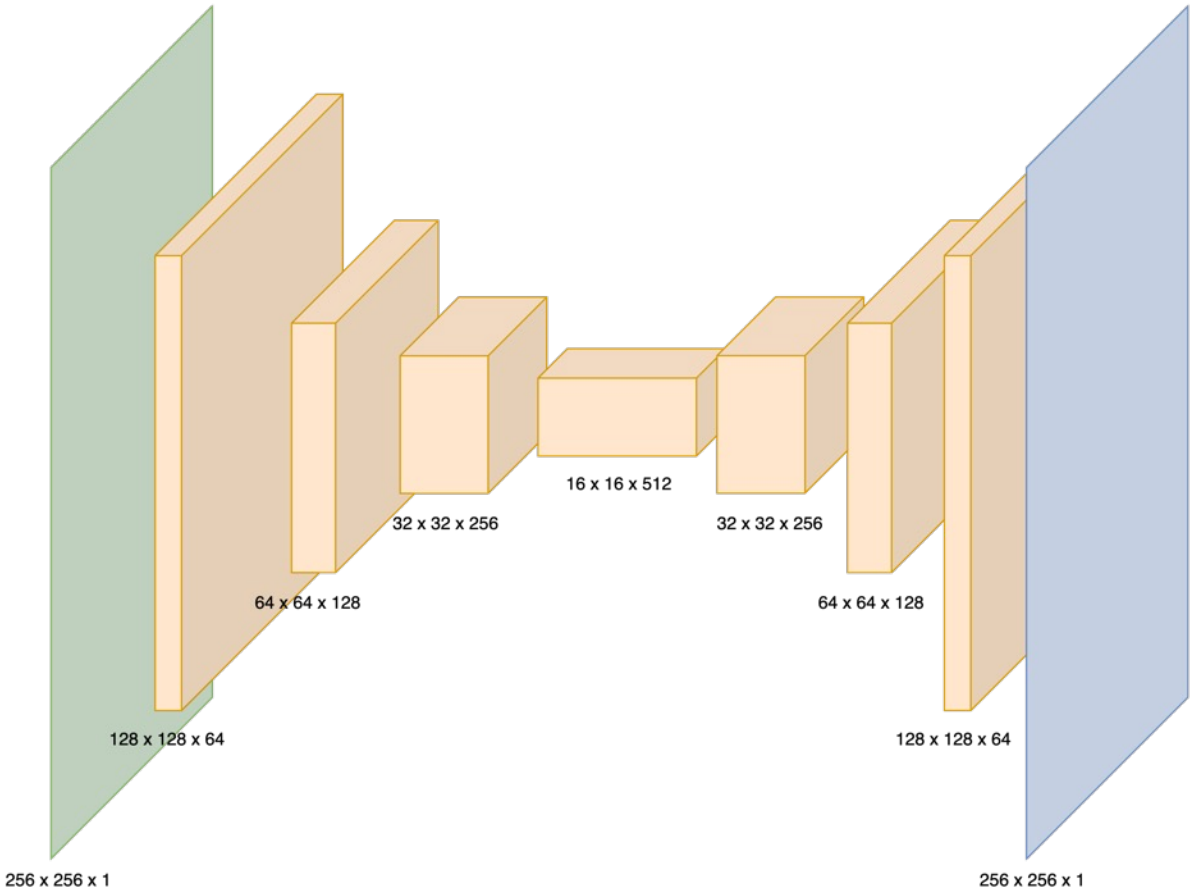


MODEL 5

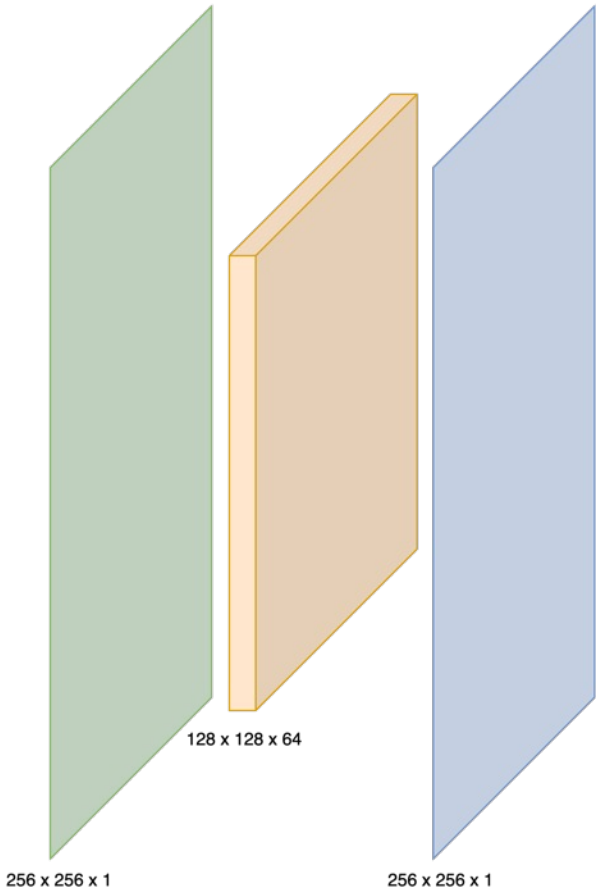


MODEL 6

kernel size 3 × 3
stride of 2
ReLU
adam solver
learning rate of 0.0001



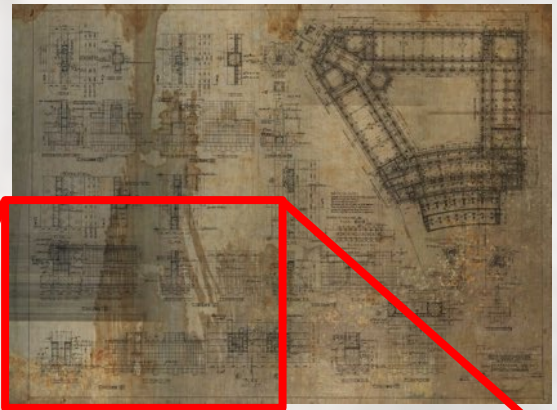
MODEL 7



MODEL 8

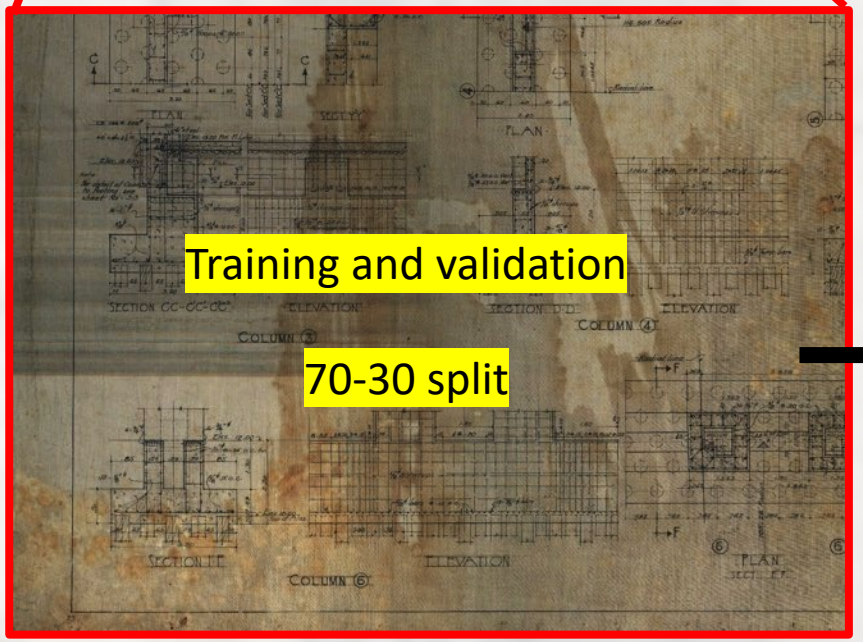
kernel size 3 × 3
stride of 2
ReLU
adam solver
learning rate of 0.0001

Creating the ideal clean version which will serve as the output of autoencoders

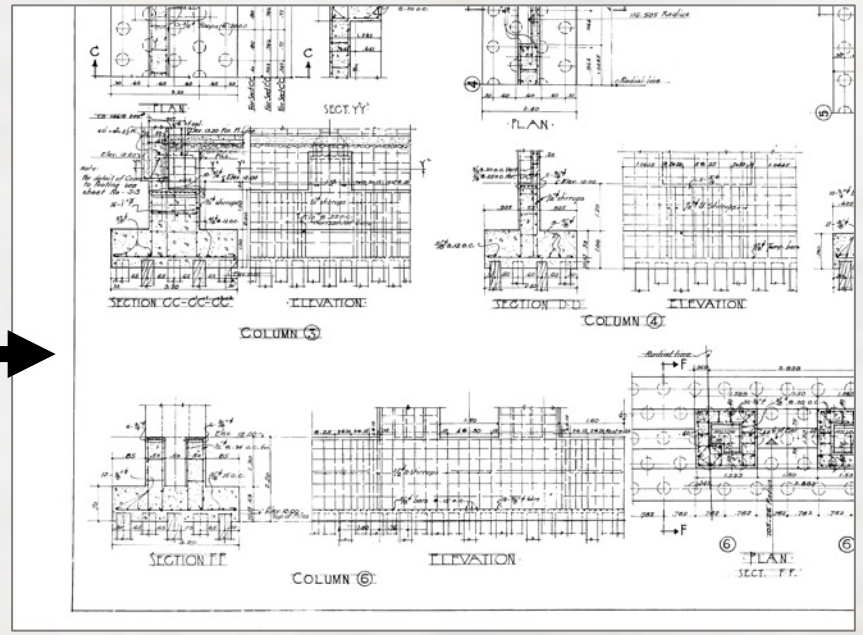


Drawing 1

Cleaning took 21 hours
in a span of 7 days

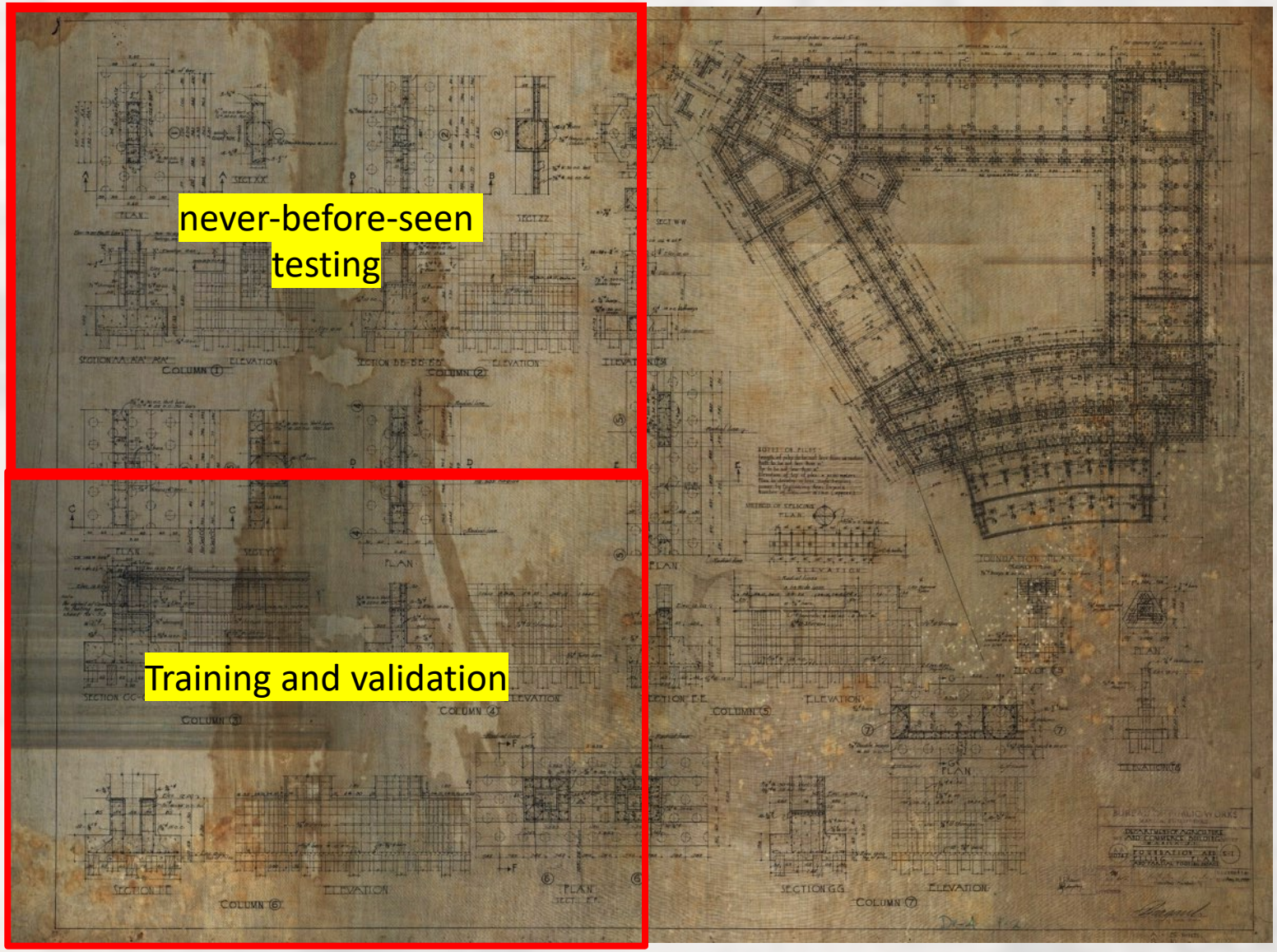


Portion of Drawing 1

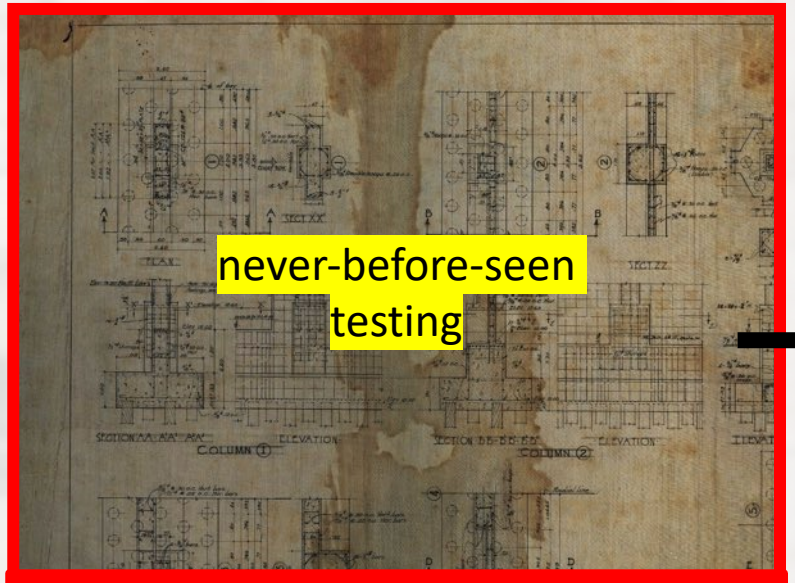


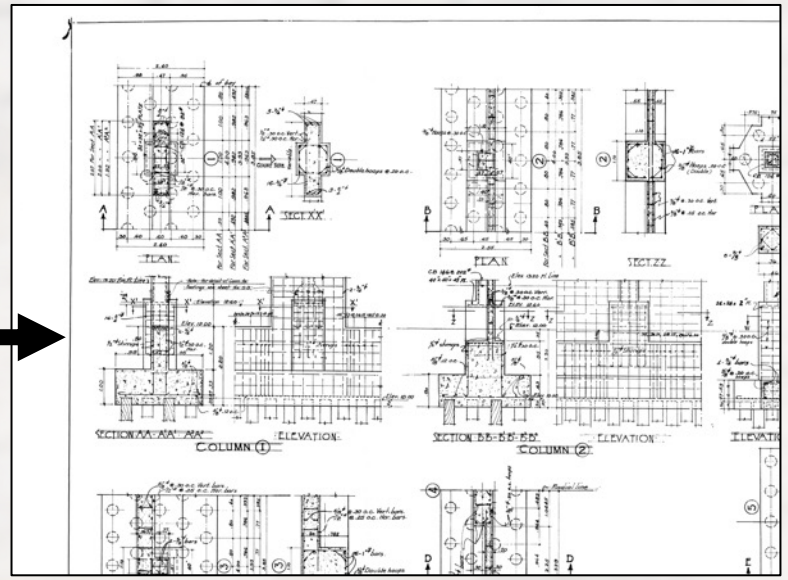
Manually cleaned portion of Drawing 1

Creating a ground-truth to assess our model



Creating a ground-truth to assess our model





ground-truth

Table 3. Metric scores for the 8 models (100 epoch) and traditional thresholding technique. The number of training images are **22,484 – 22,644.**

Input size	Technique	Precision	Recall	F1score	IOU	PSNR
32 x 32	**MODEL 1***	0.9687	0.9679	0.9683	0.9385	23.99
	MODEL 2*	0.9702	0.9788	0.9745	0.9502	24.93
64 x 64	**MODEL 3***	0.9626	0.9597	0.9612	0.9253	23.11
	MODEL 4*	0.9727	0.9772	0.975	0.9511	25.02
128 x 128	**MODEL 5****	0.9679	0.9629	0.9654	0.9331	23.62
	MODEL 6**	0.9742	0.9799	0.977	0.9551	25.4
256 x 256	**MODEL 7****	0.9568	0.9664	0.9616	0.926	23.13
	MODEL 8**	0.9772	0.9765	**0.9768**	**0.9547**	**25.37**
Whole image	Traditional thresholding	0.9286	0.9745	0.9510	0.9083	22.07

shallow

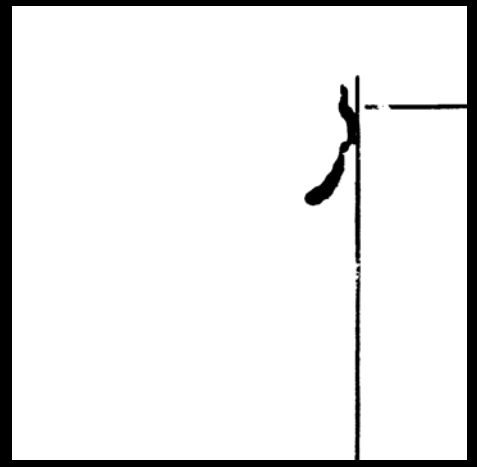
*Google Colab
**Google Colab Pro

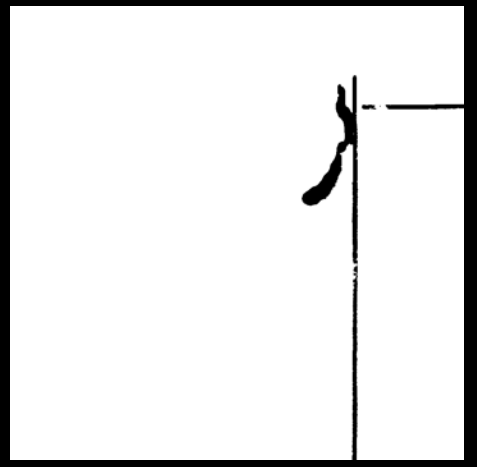
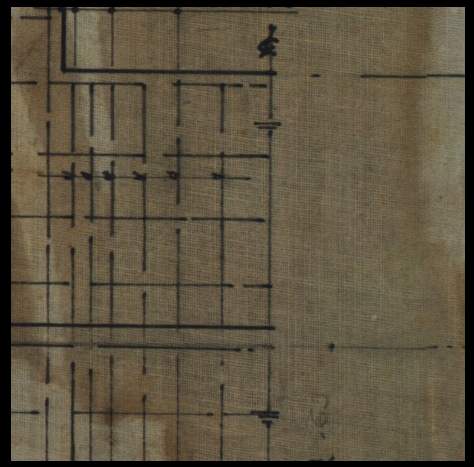
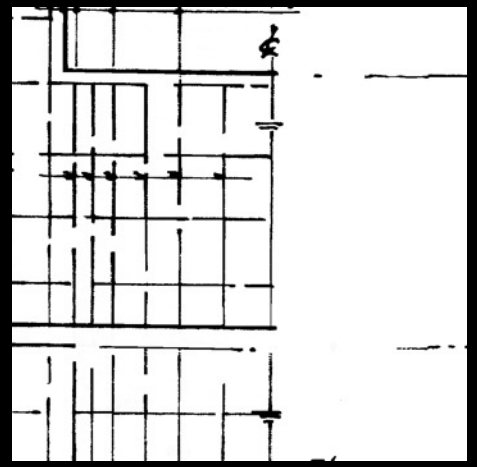
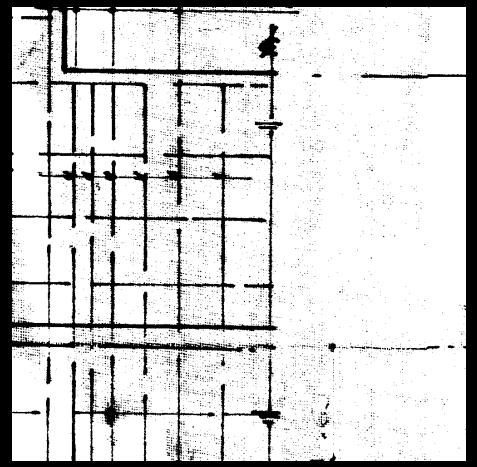
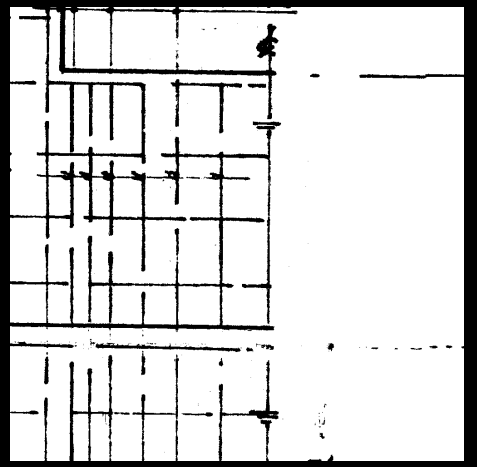
Input

Ground truth

Traditional thresholding

Ours

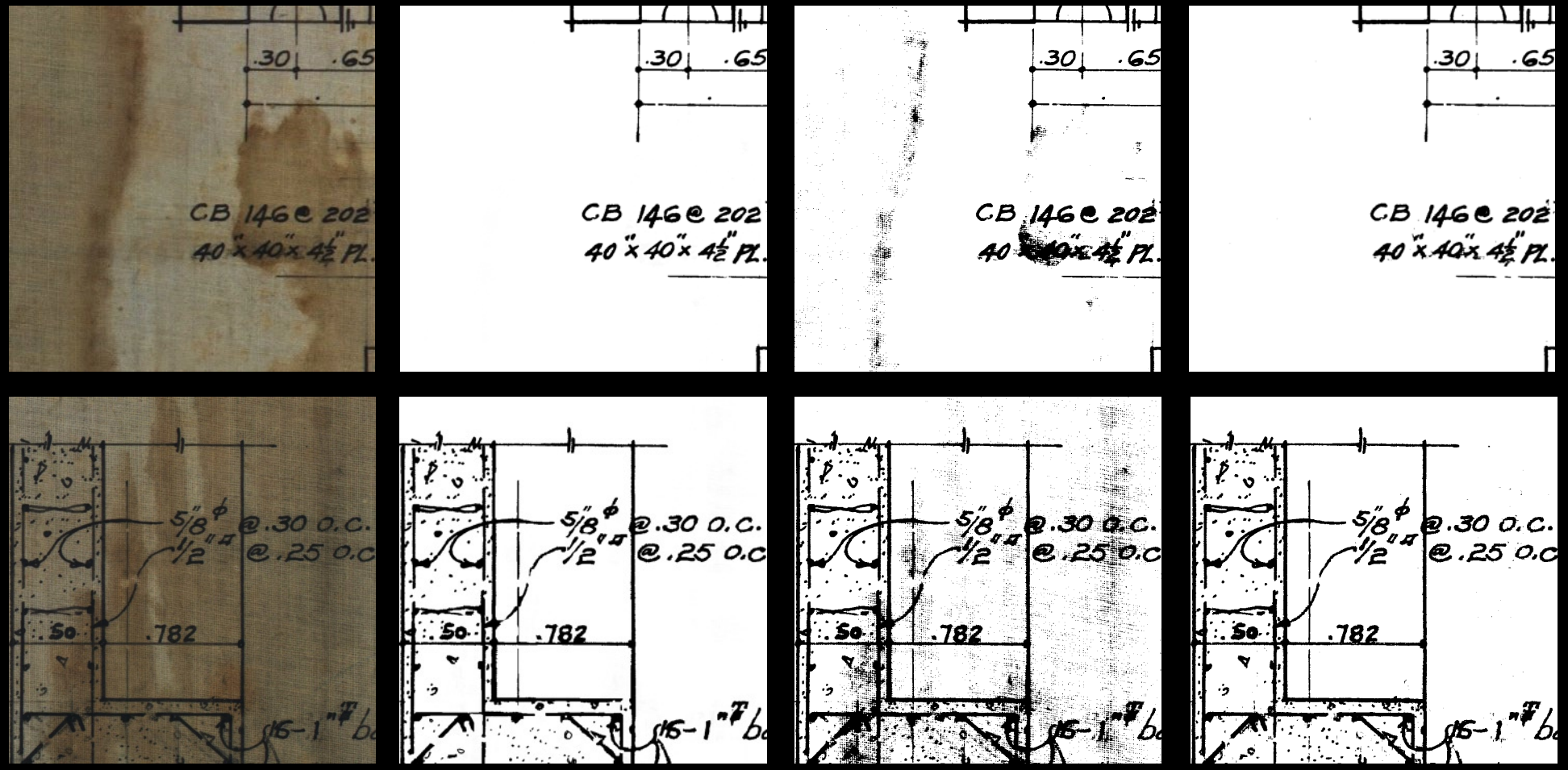






Input

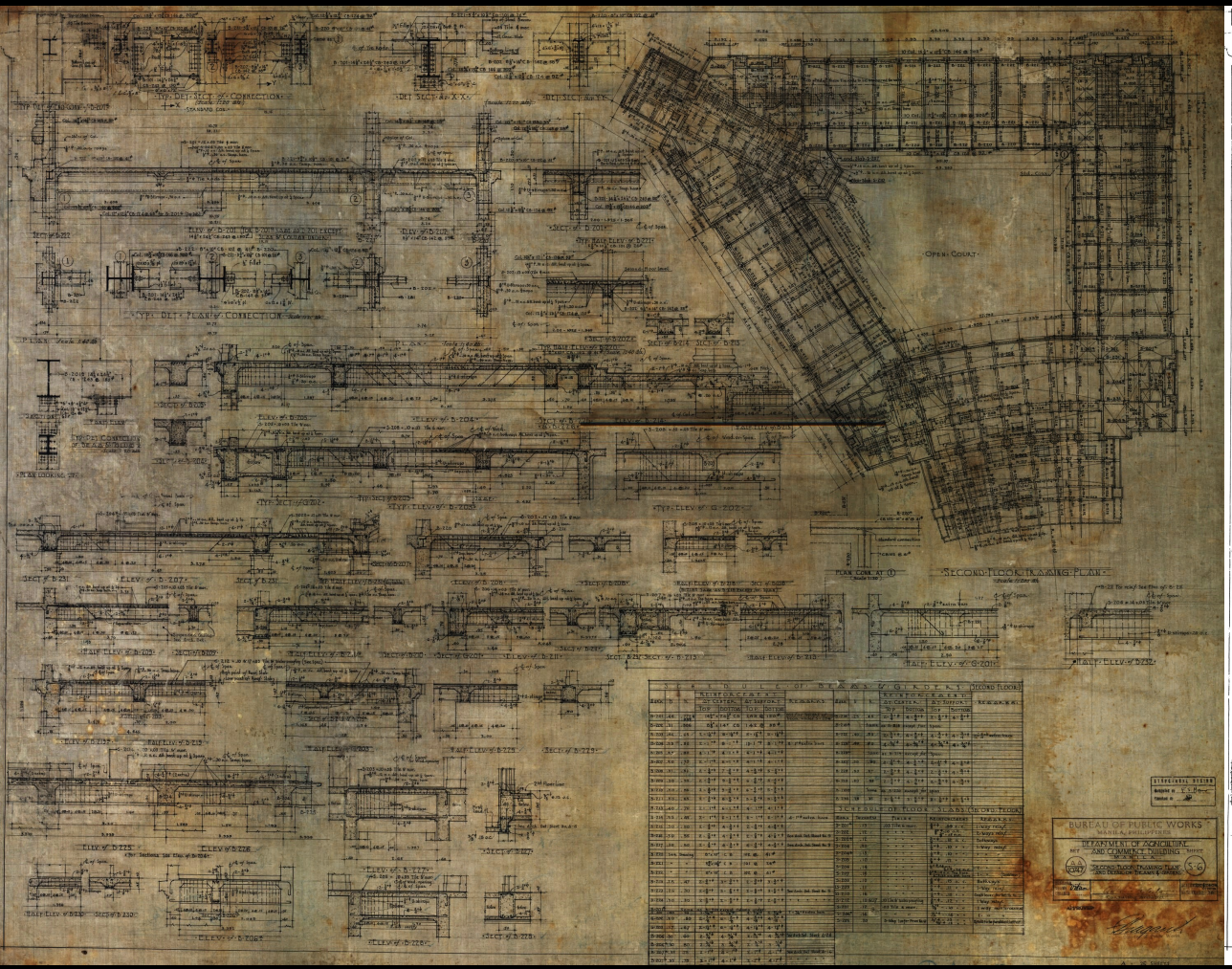
Ground truth

Traditional thresholding

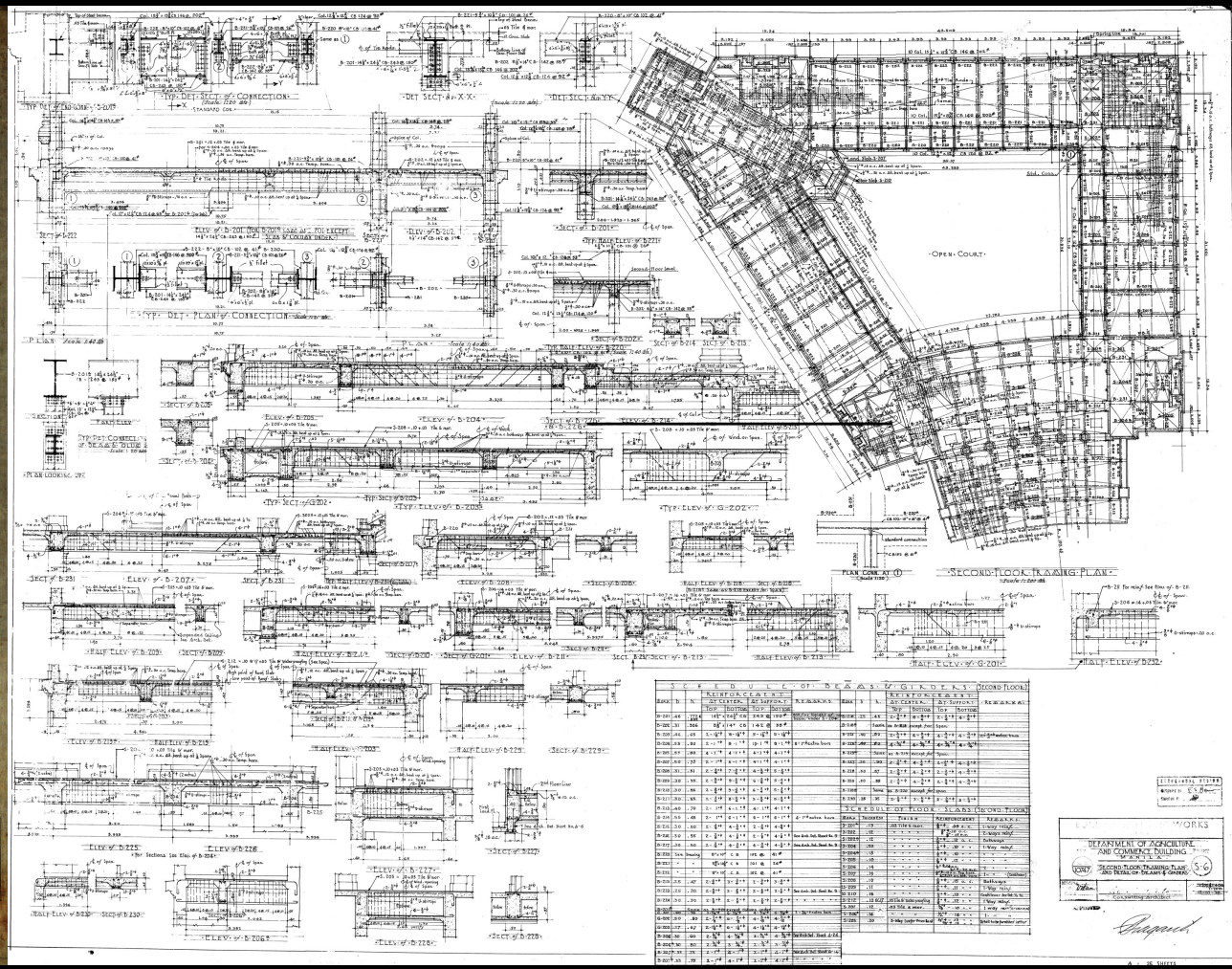
Ours



Architectural drawing 3

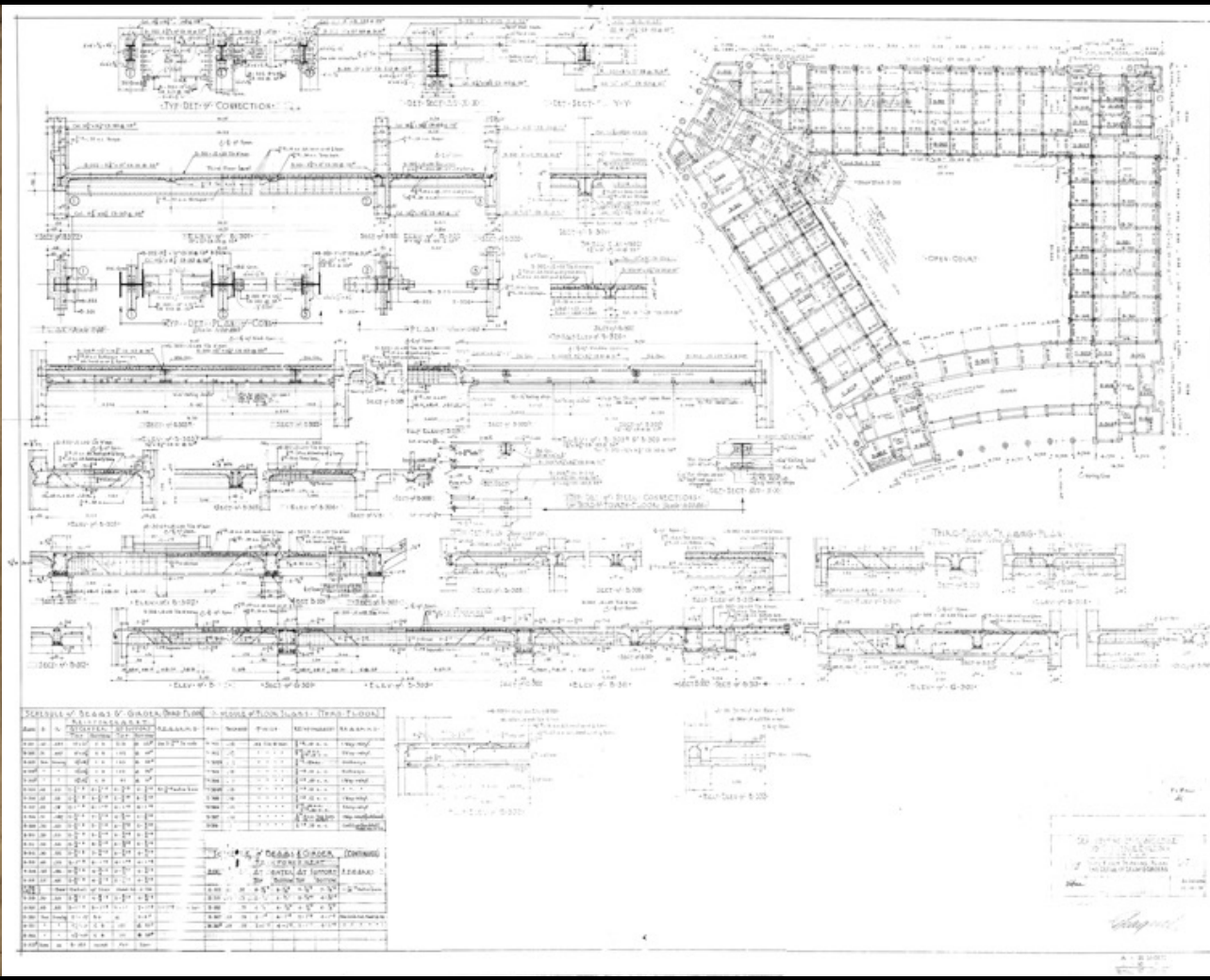


Ours



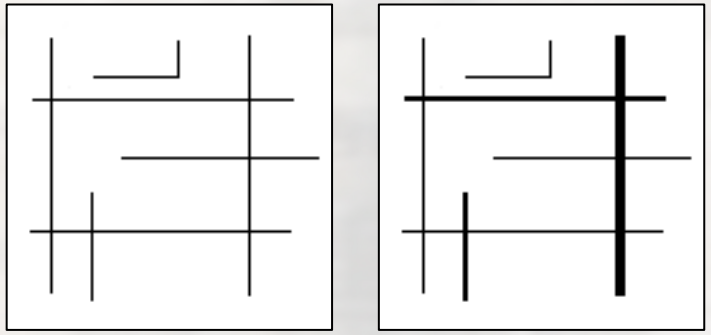
Architectural drawing 2

Ours

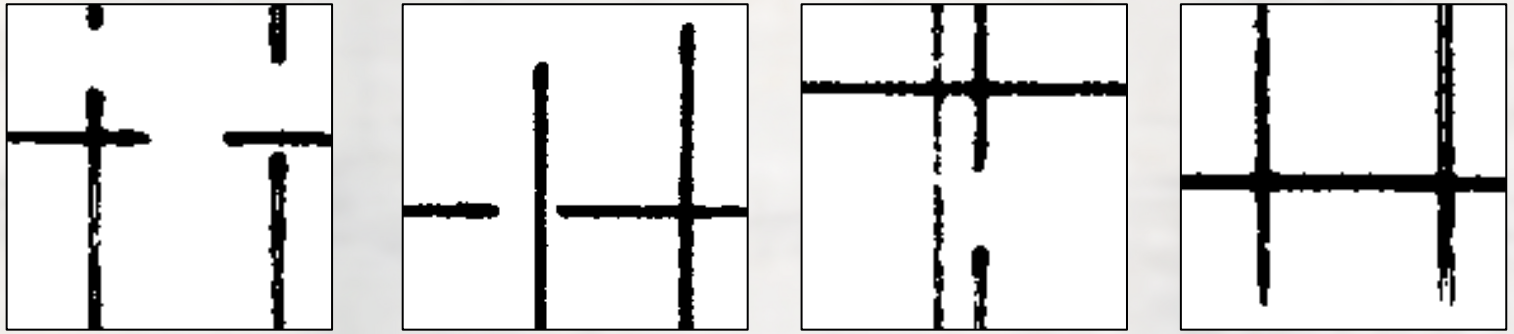


STAGE 2: VECTORIZATION

Part 0: Detecting corners from synthetic line images

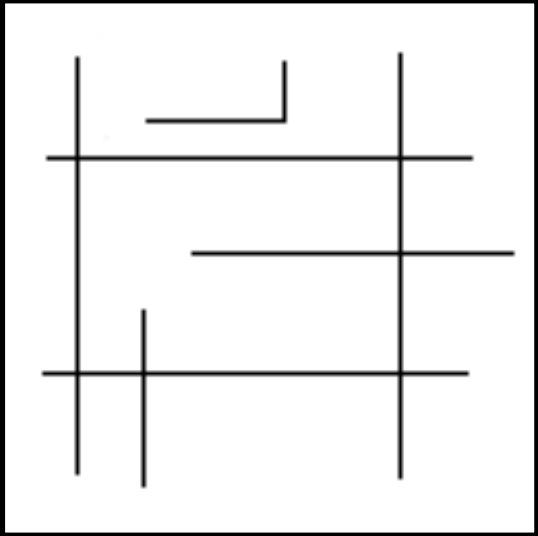


Part 1: Detecting corners from actual image (Shi-Tomasi vs Template matching)

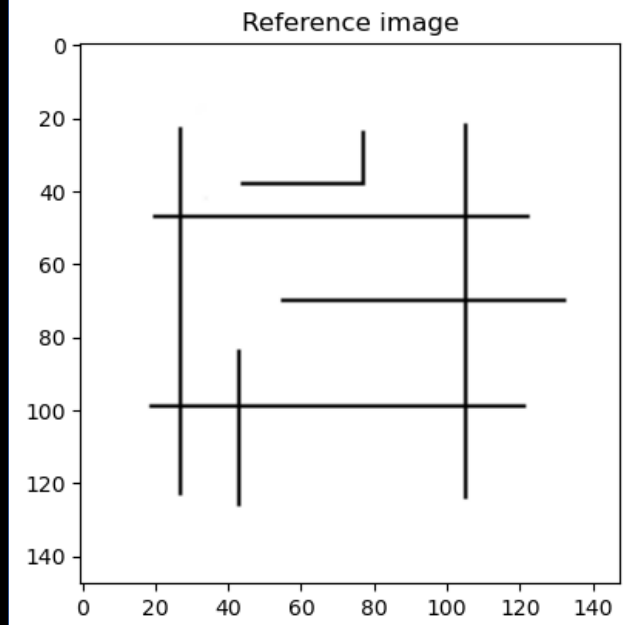


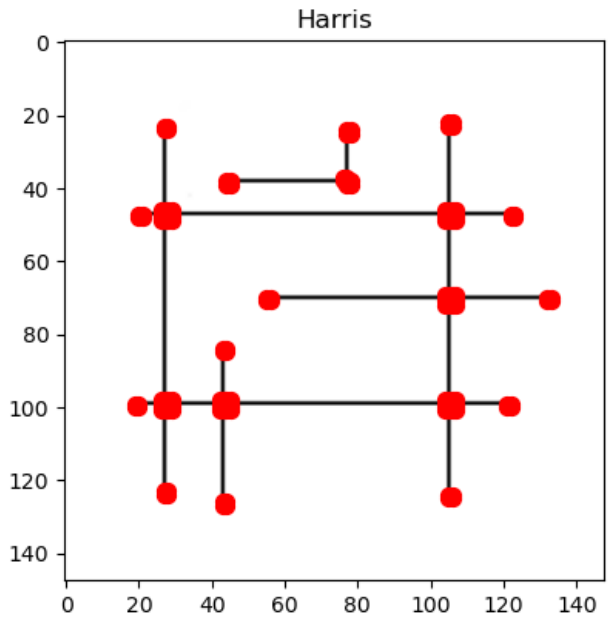
Part 2: Line coordinate tracing

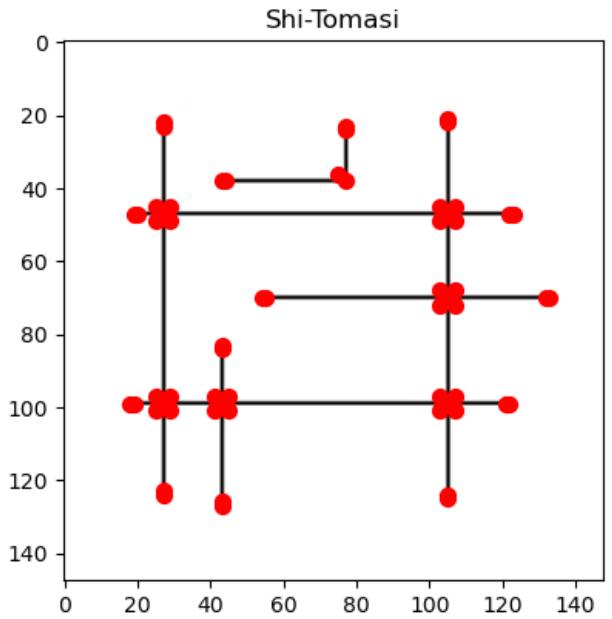
Part 0: Detecting corners from synthetic line images

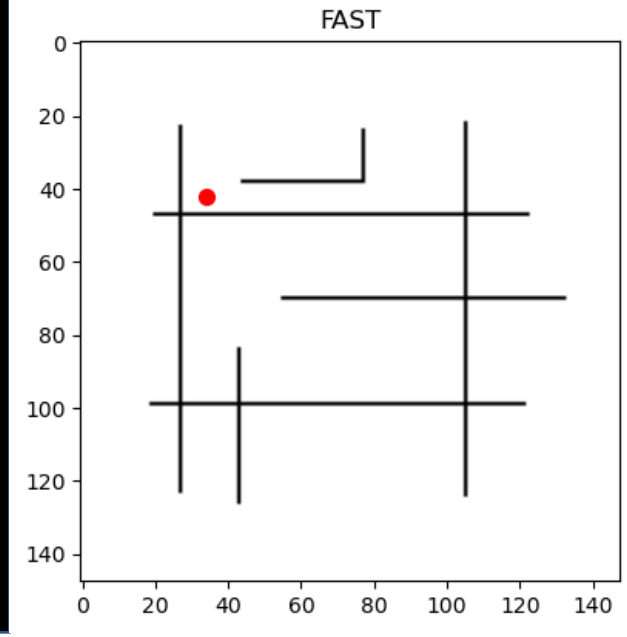


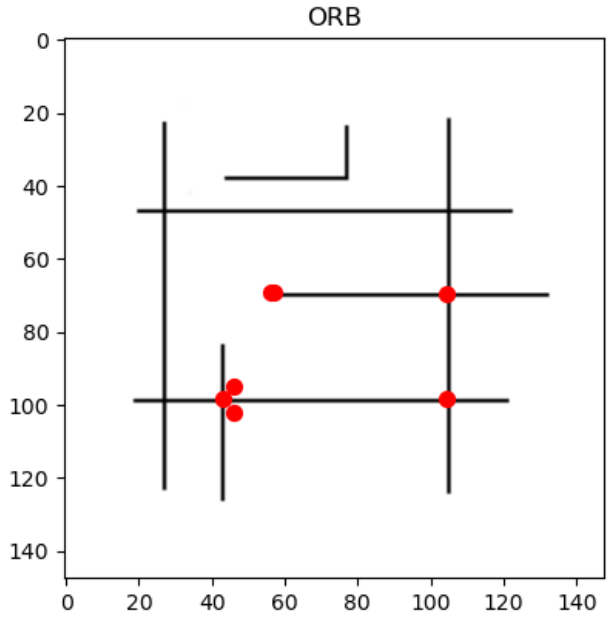
Shi-Tomasi:
max_corners = 100
quality_level = 0.01
min_distance = 5

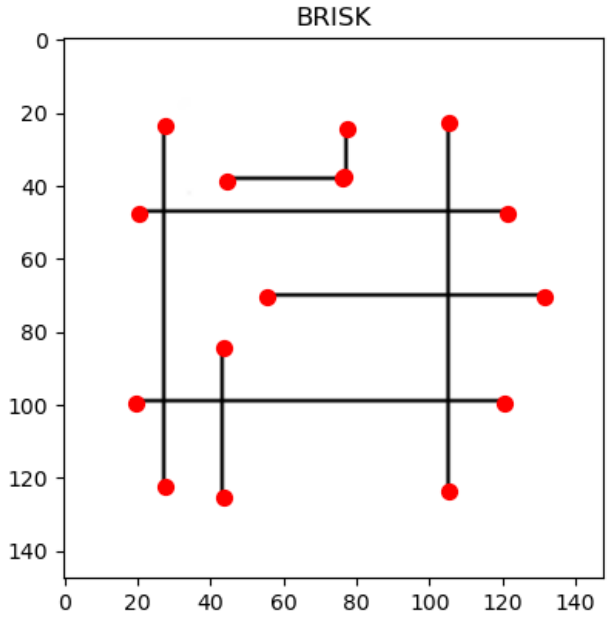












Part 0: Detecting corners from synthetic line images

1.Harris Corner Detection:

- Measures corner response based on the intensity variations in all directions.
- It is rotation invariant.

2.Shi-Tomasi Corner Detection (Good Features to Track):

- A modification of Harris corner detector with improved performance.
- It selects corners based on the minimum eigenvalue of the covariance matrix.

3.FAST (Features from Accelerated Segment Test):

- A corner detection algorithm that is computationally efficient.
- It classifies pixels as corners based on intensity comparisons..

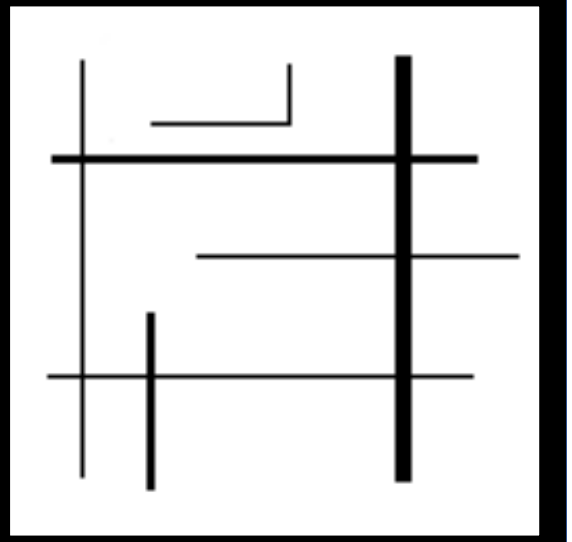
4.ORB (Oriented FAST and Rotated BRIEF):

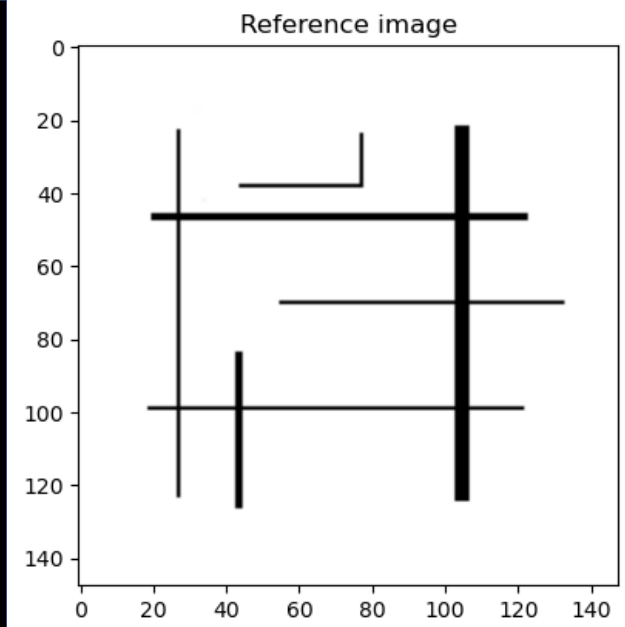
- Combines keypoint detection and feature description.
- Uses a combination of FAST for keypoint detection and BRIEF for feature description..

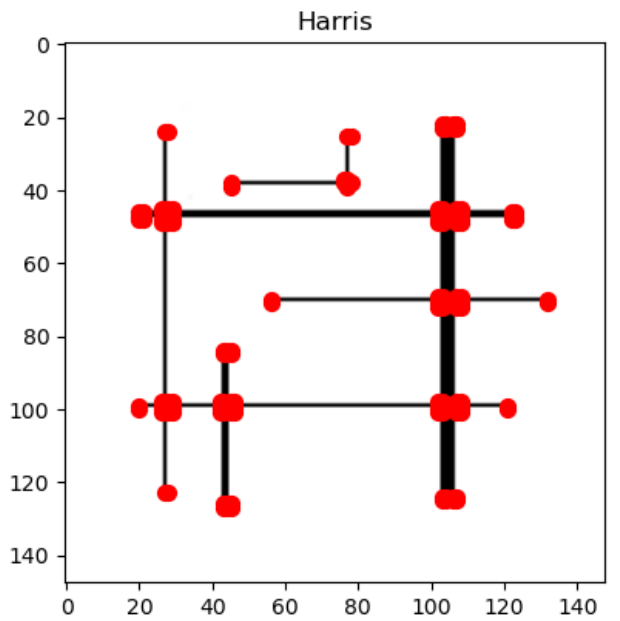
5.BRISK (Binary Robust Invariant Scalable Keypoints):

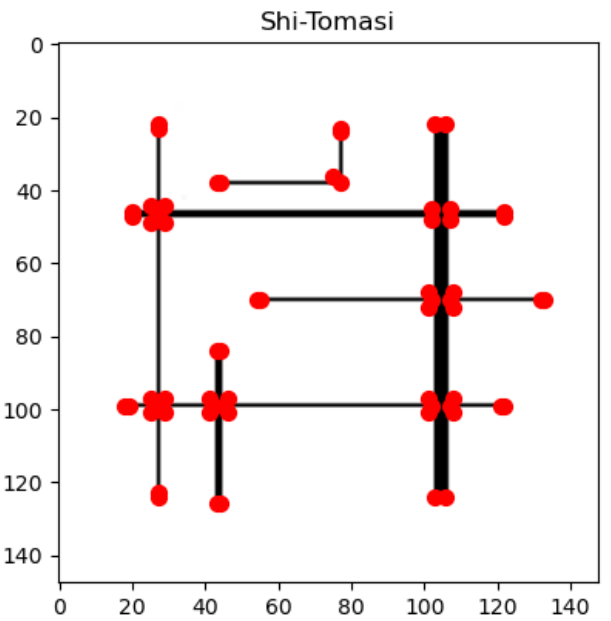
- Employs a scale-space pyramid to create a scale-invariant detector.
- Utilizes binary tests for feature description..

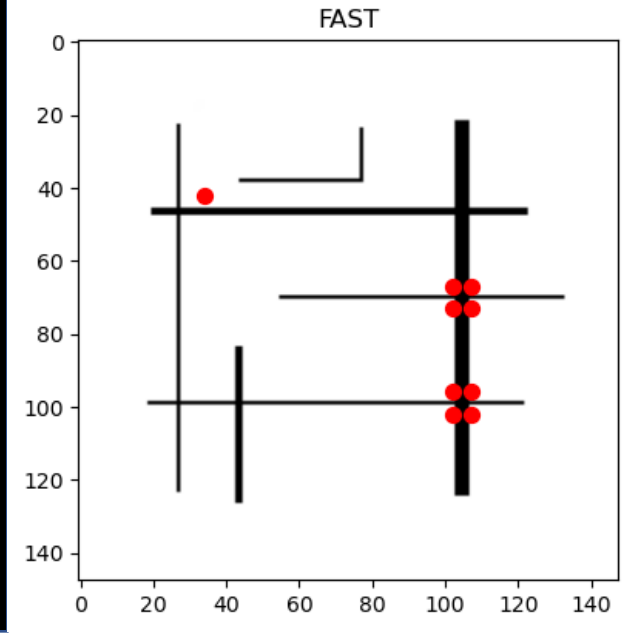
Part 0: Detecting corners from synthetic line images

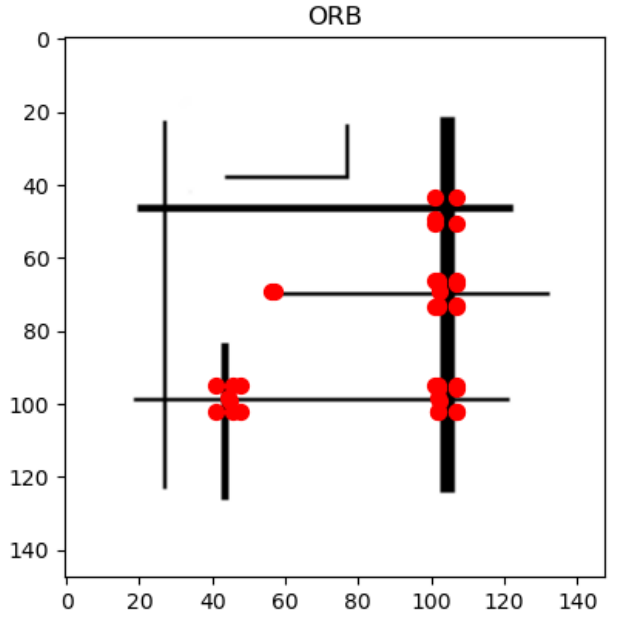


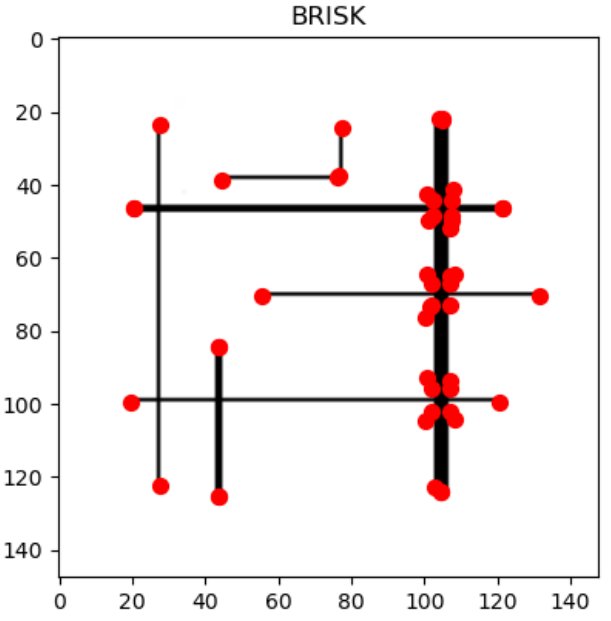




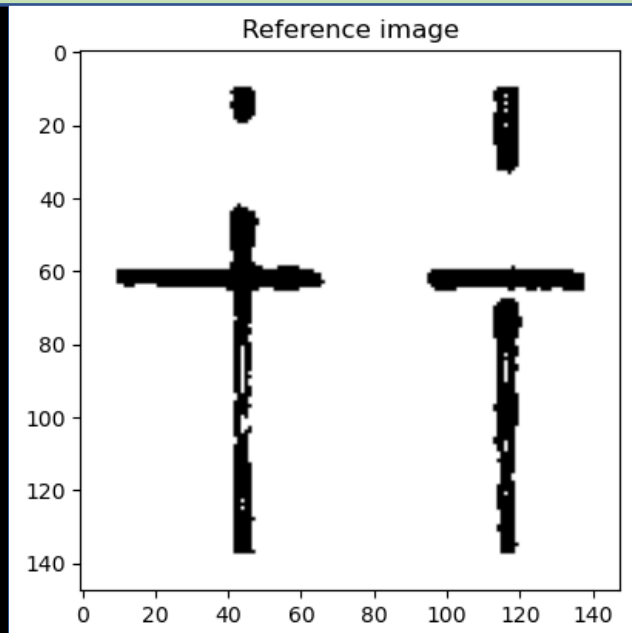


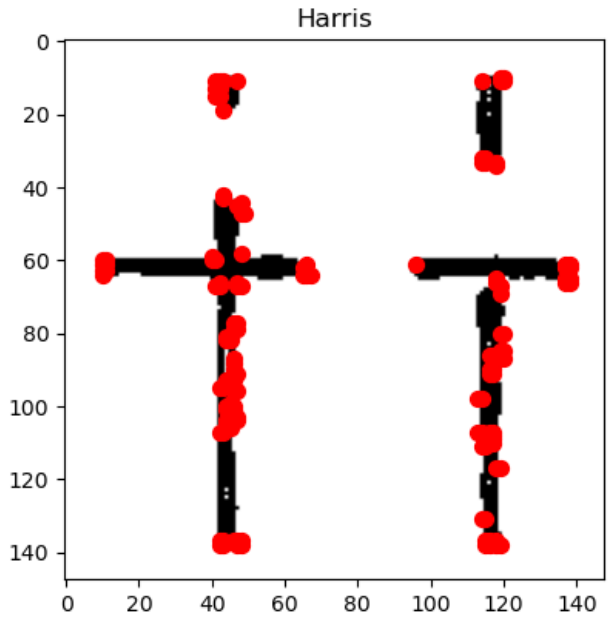


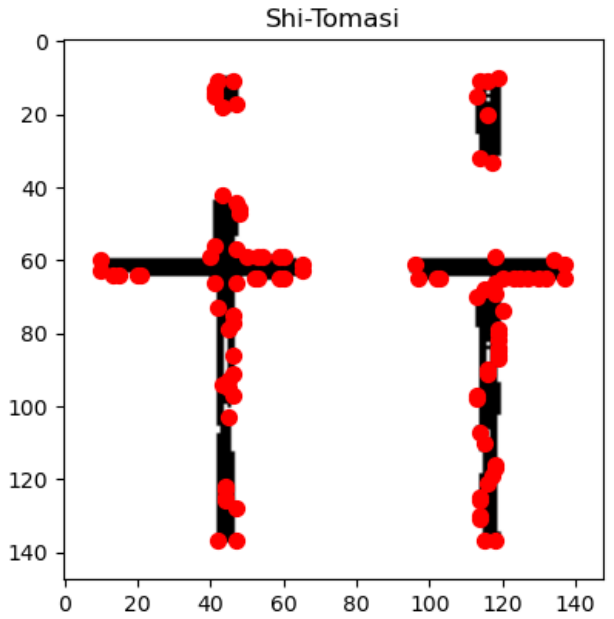


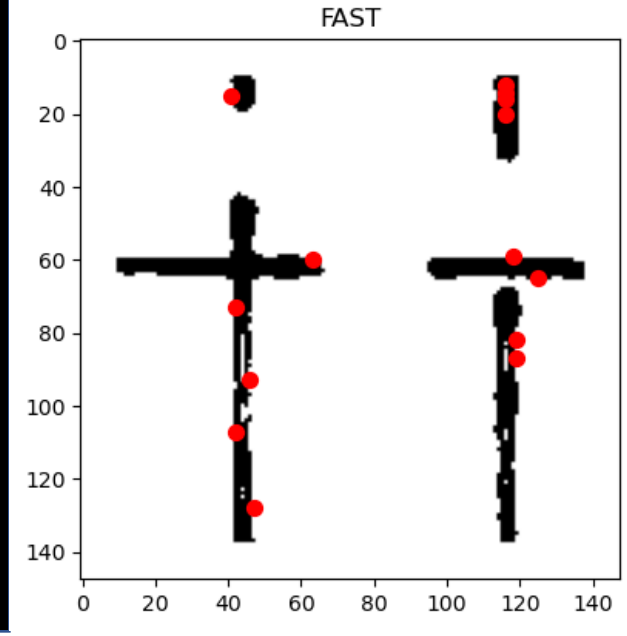


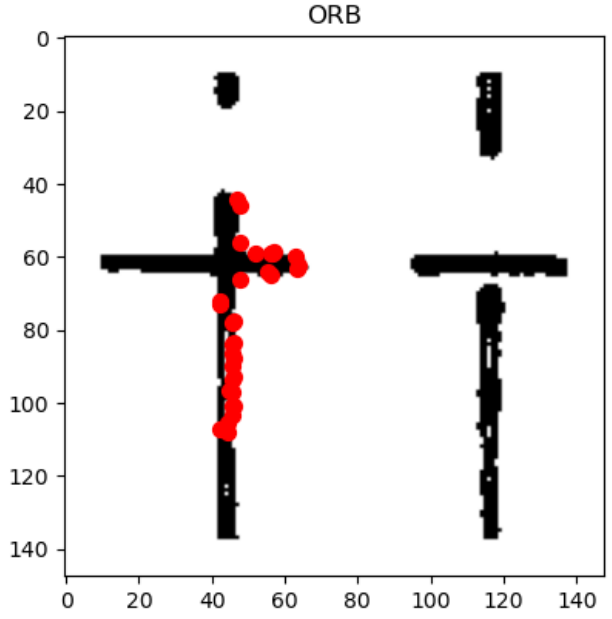
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

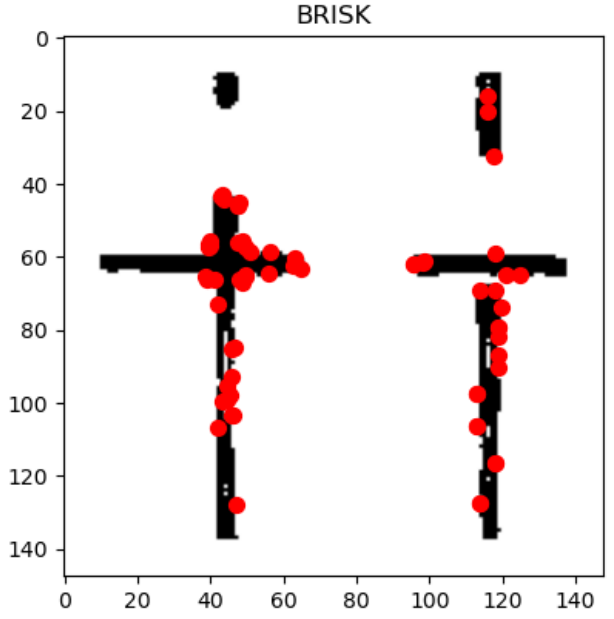






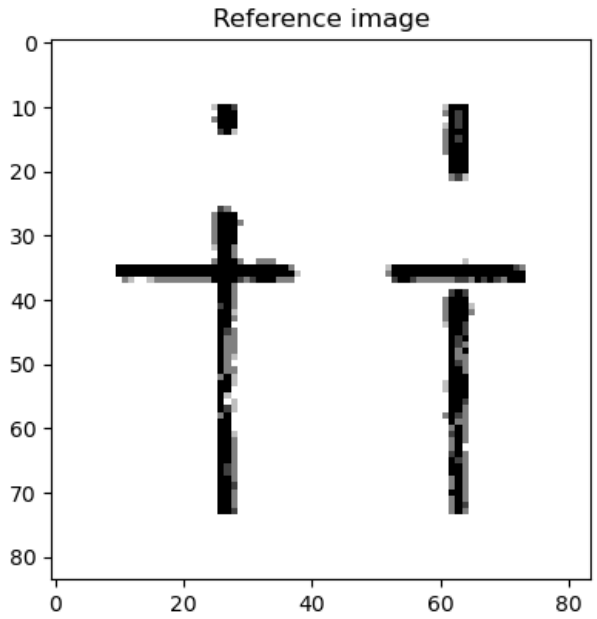


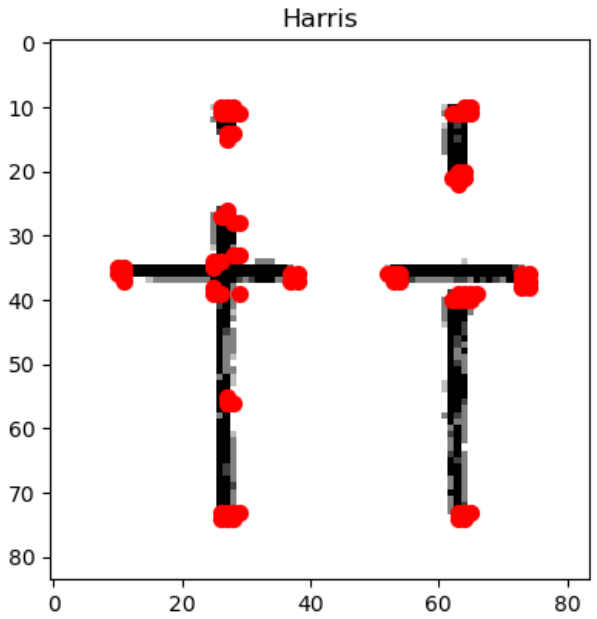


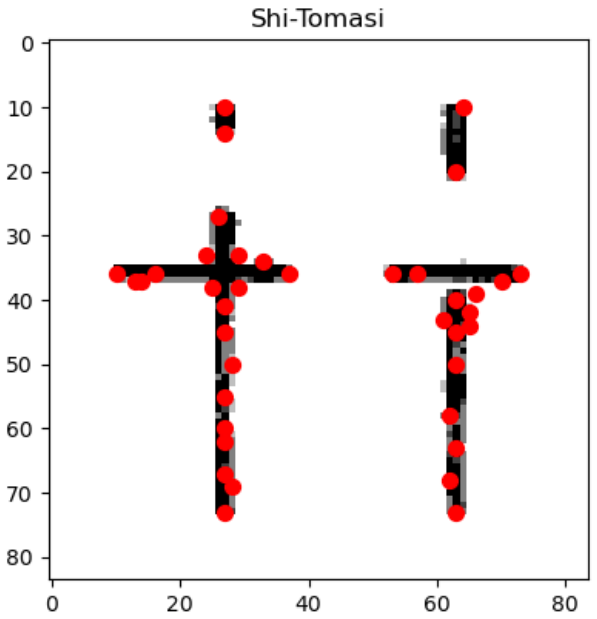


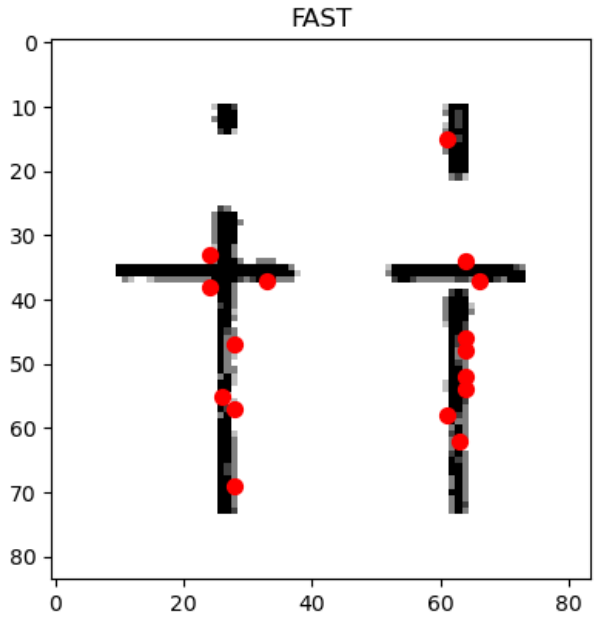
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

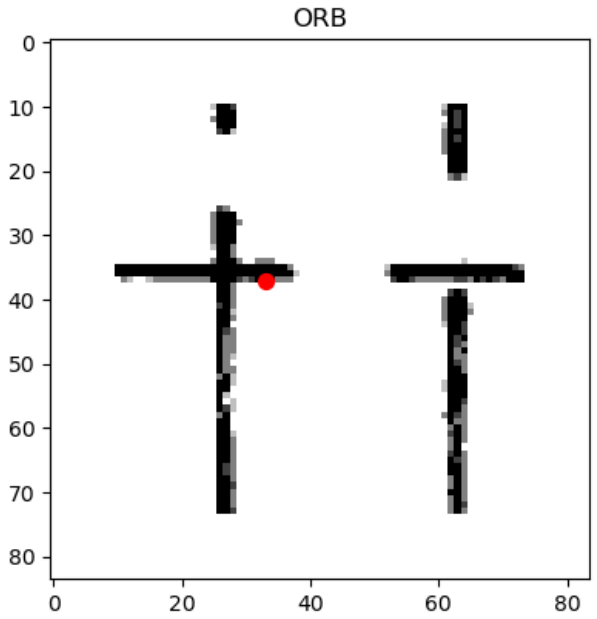
Resized to
64 by 64

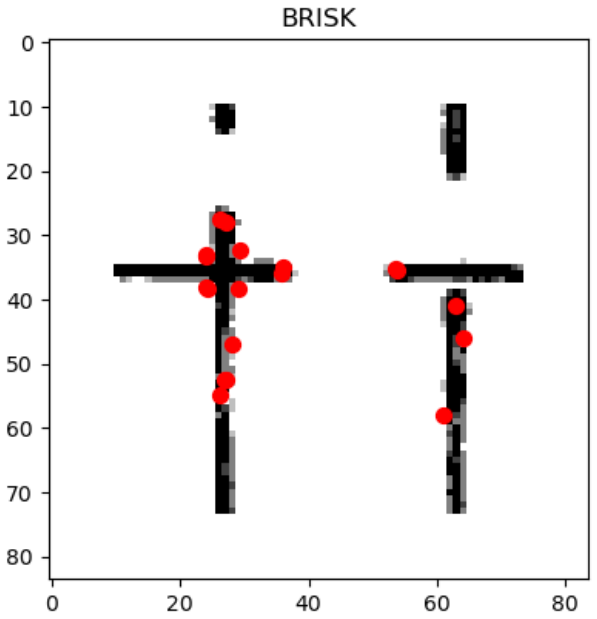






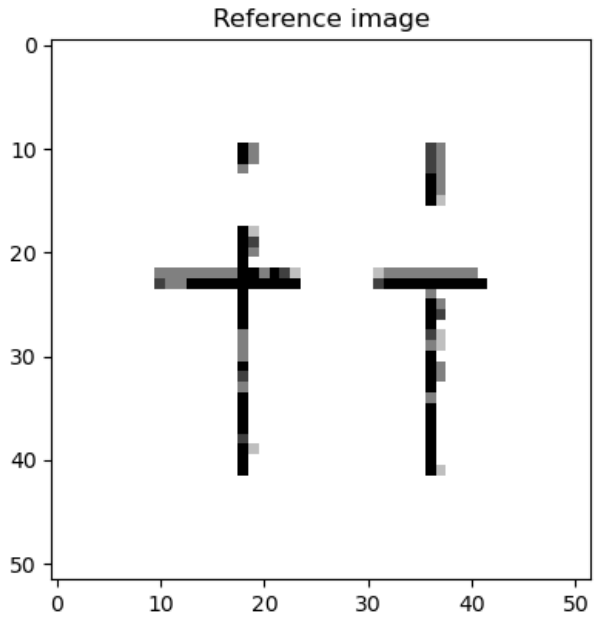


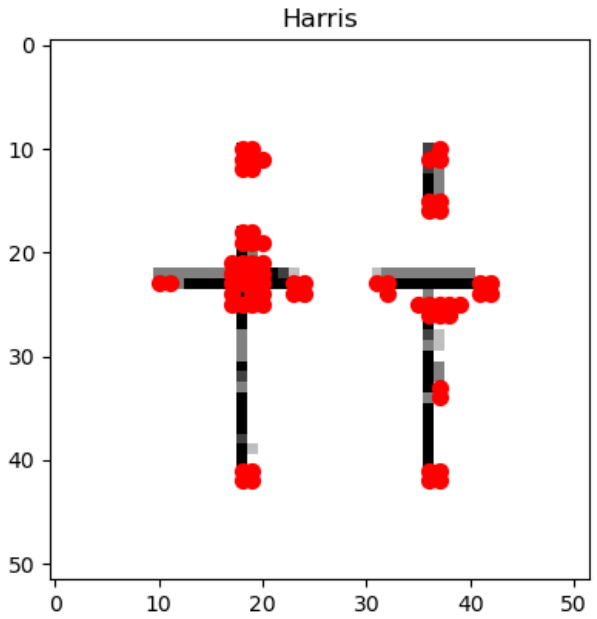


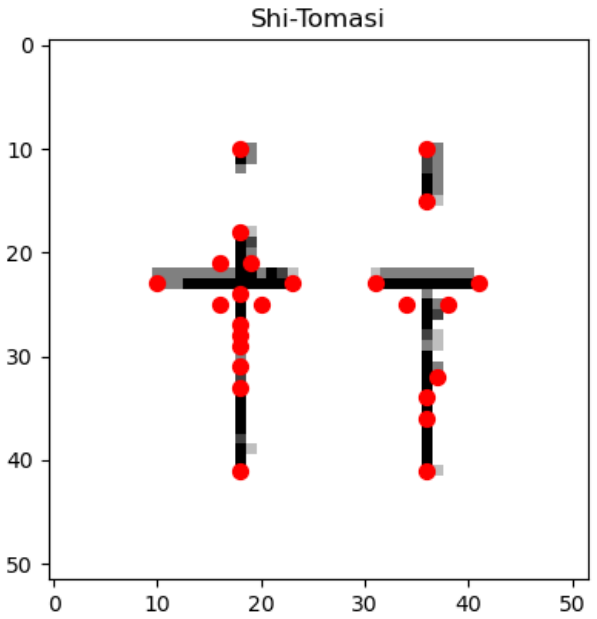


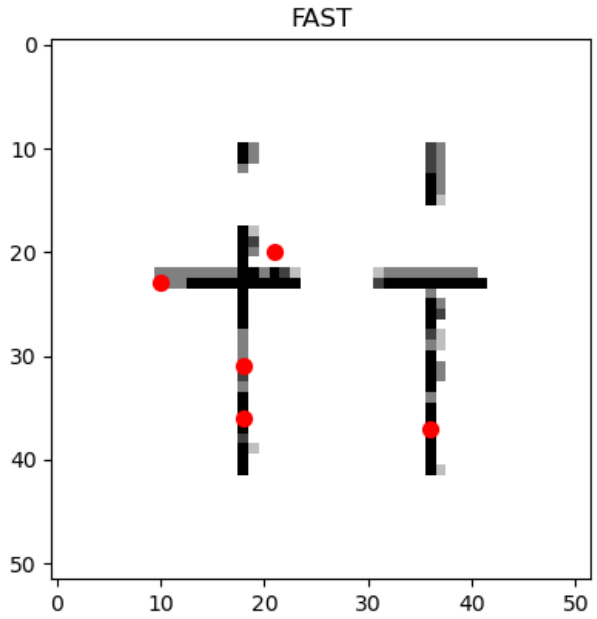
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

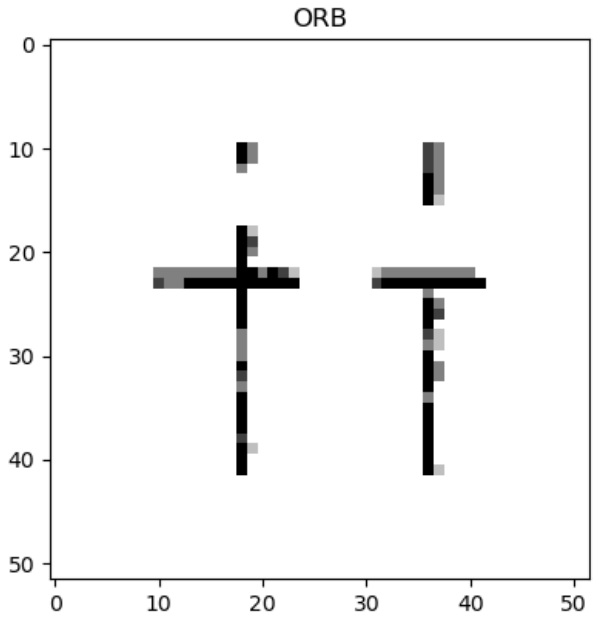
Resized to
32 by 32

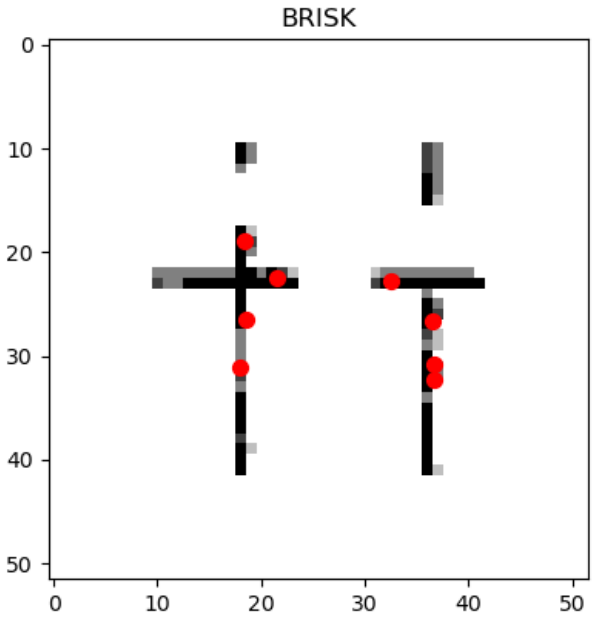






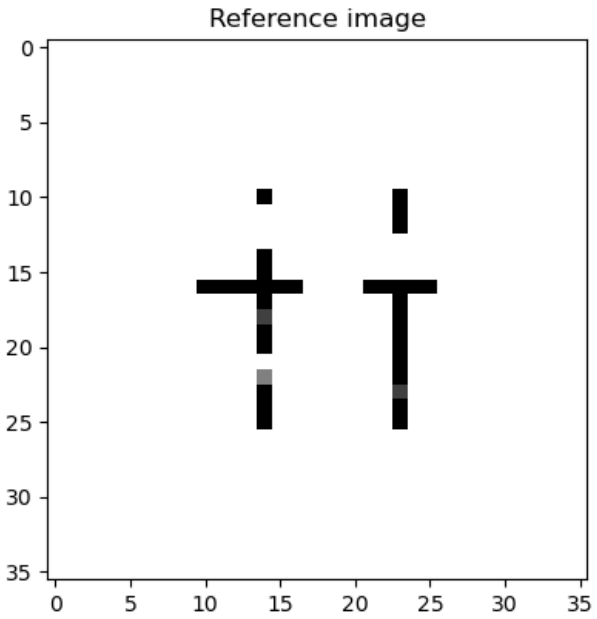


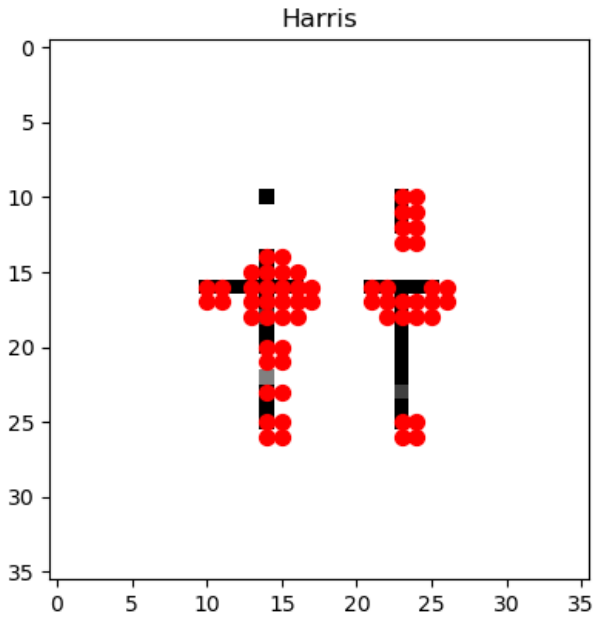


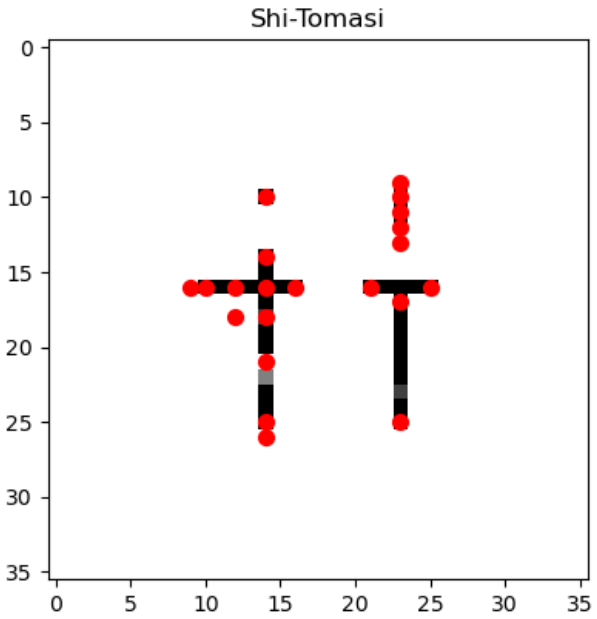


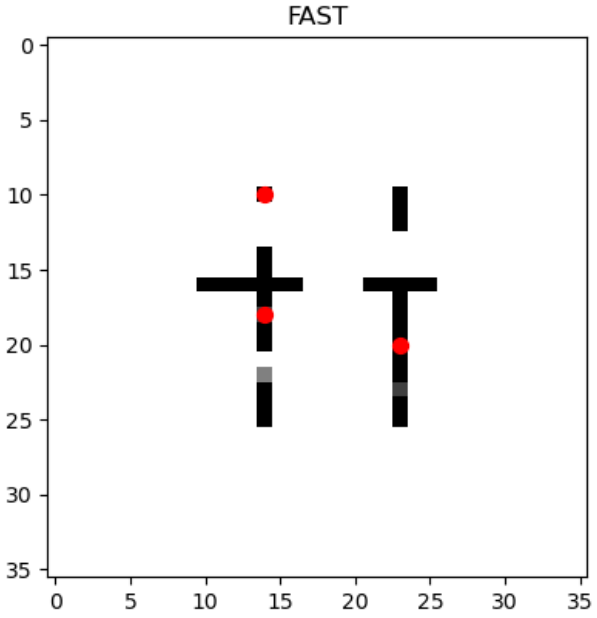
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

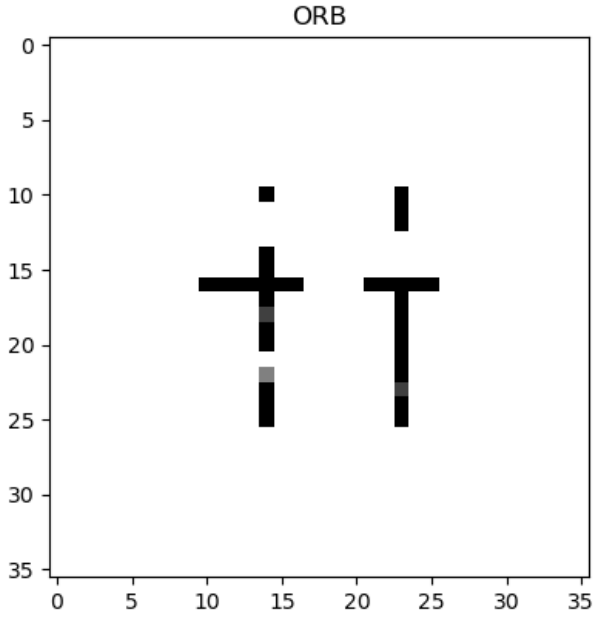
Resized to
16 by 16

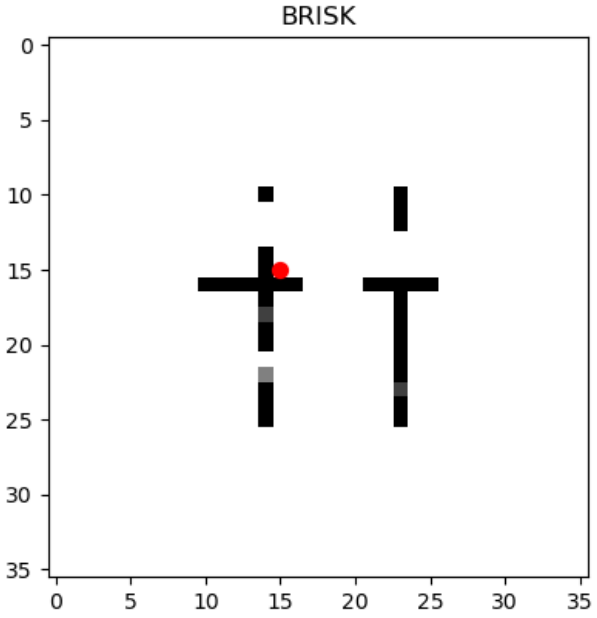






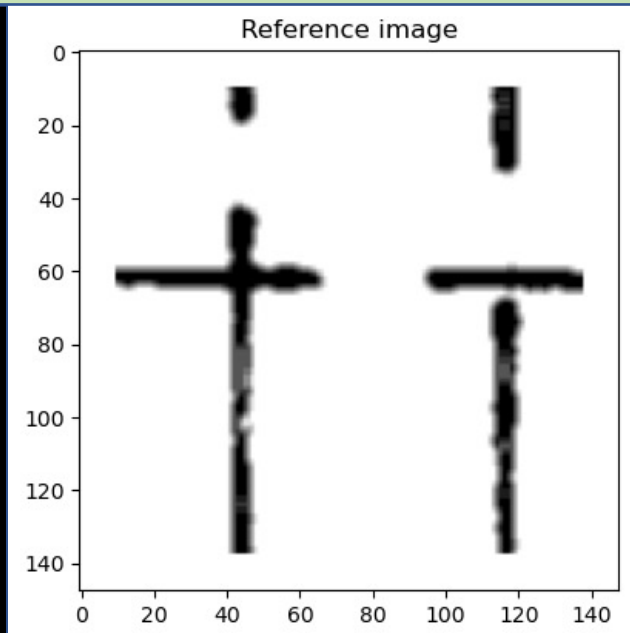


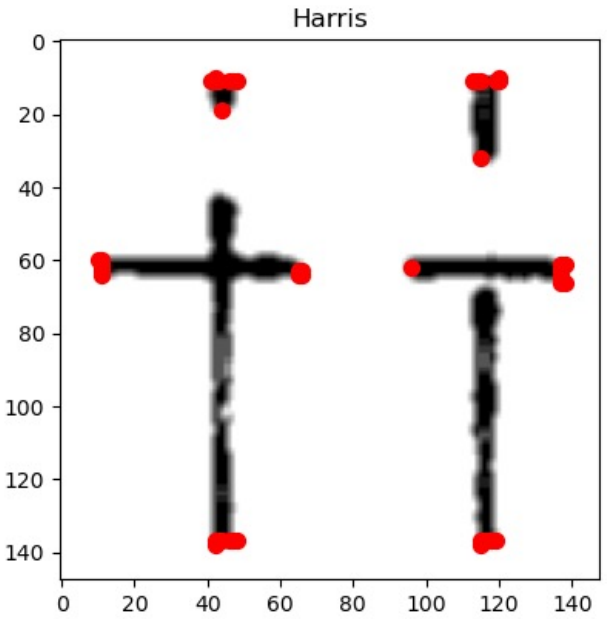


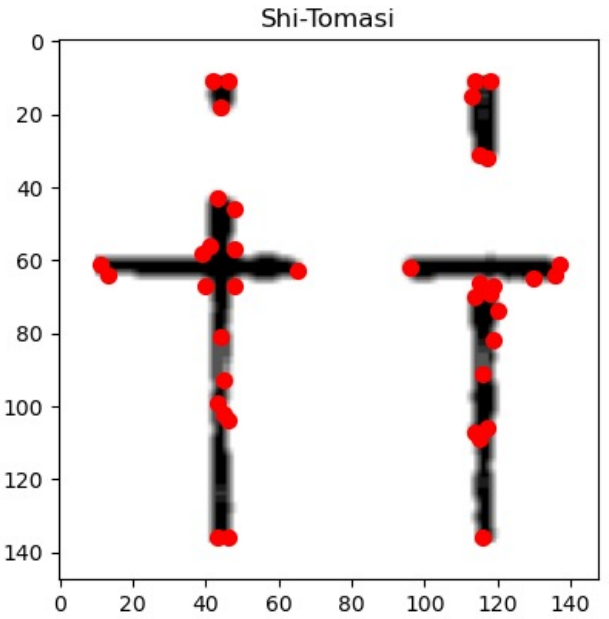


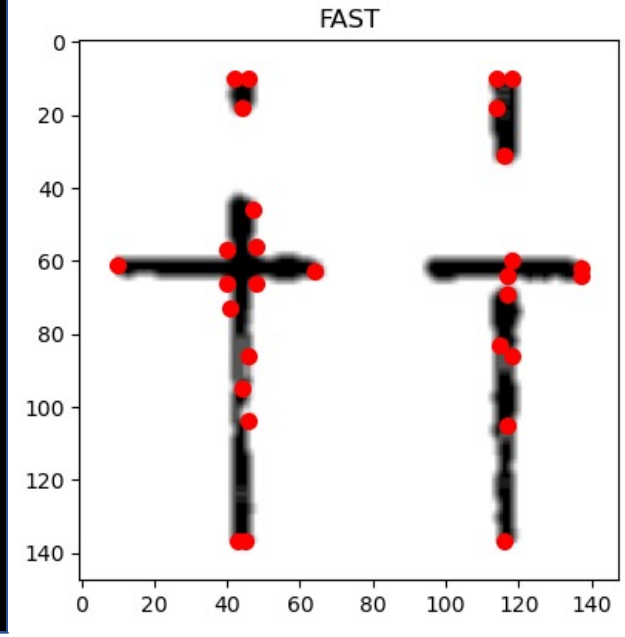
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

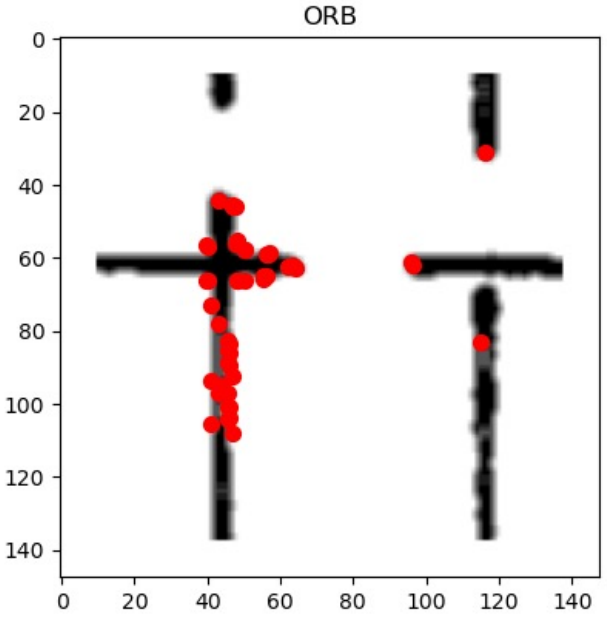
Gaussian blur (kernel 3 by 3)

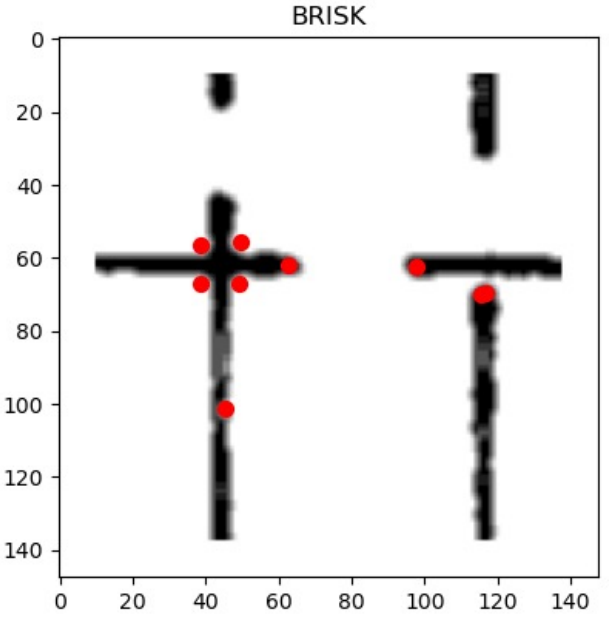






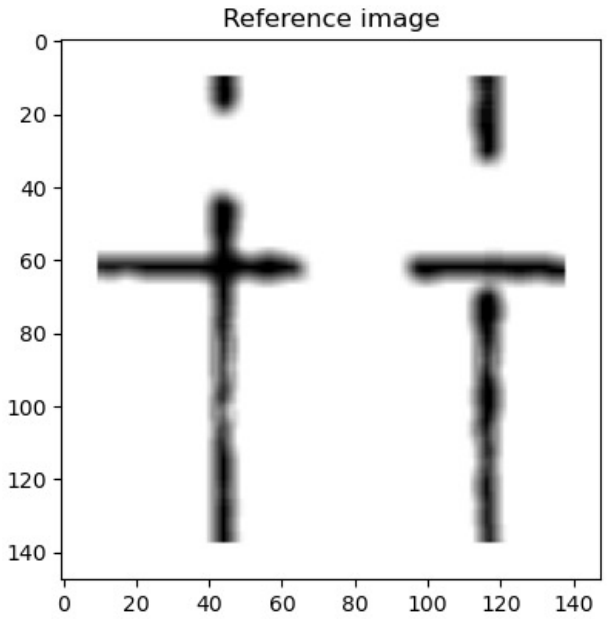


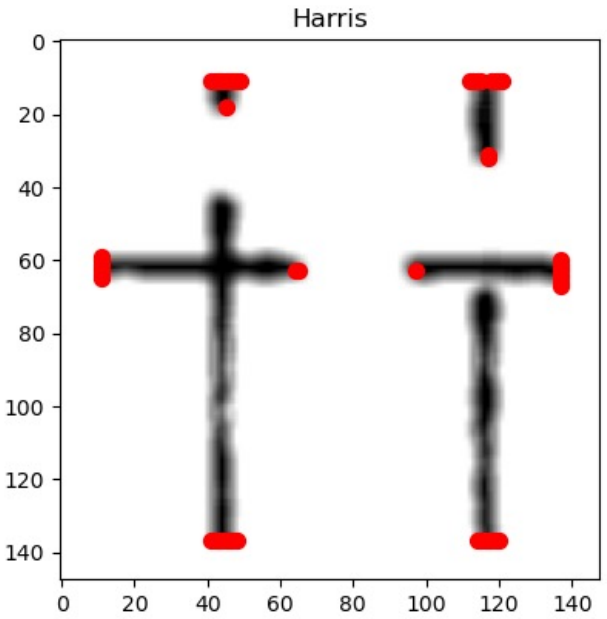


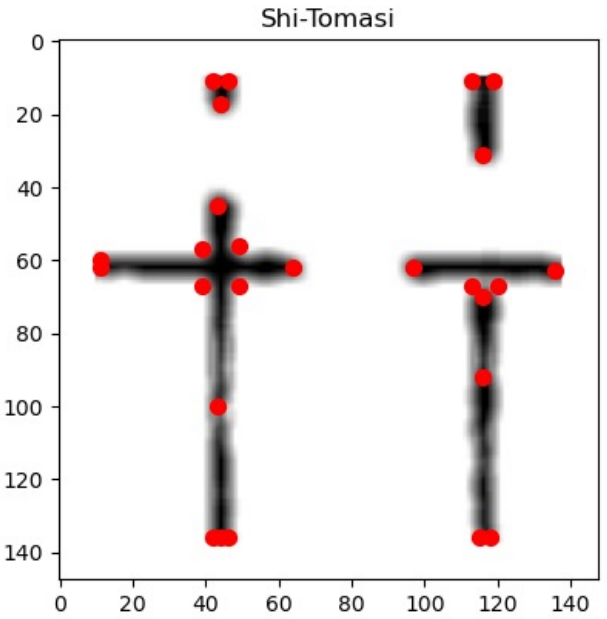


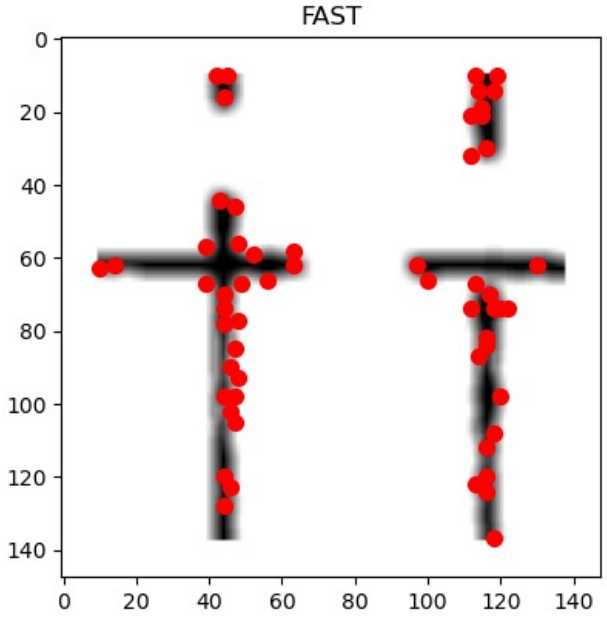
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

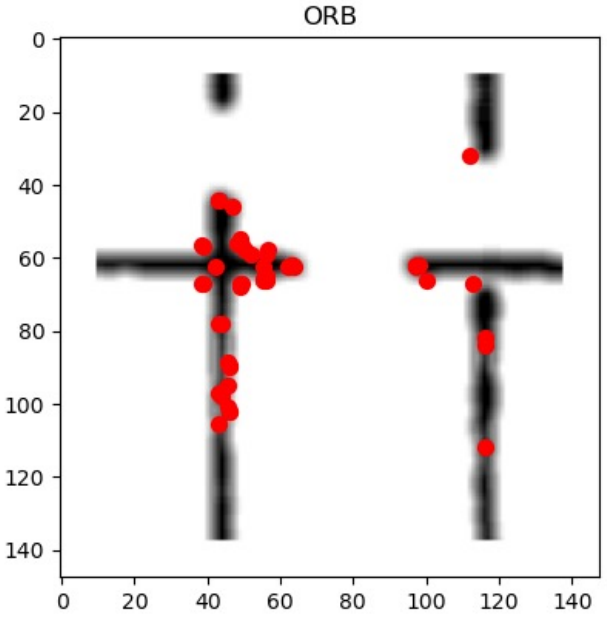
Gaussian blur (kernel 5 by 5)

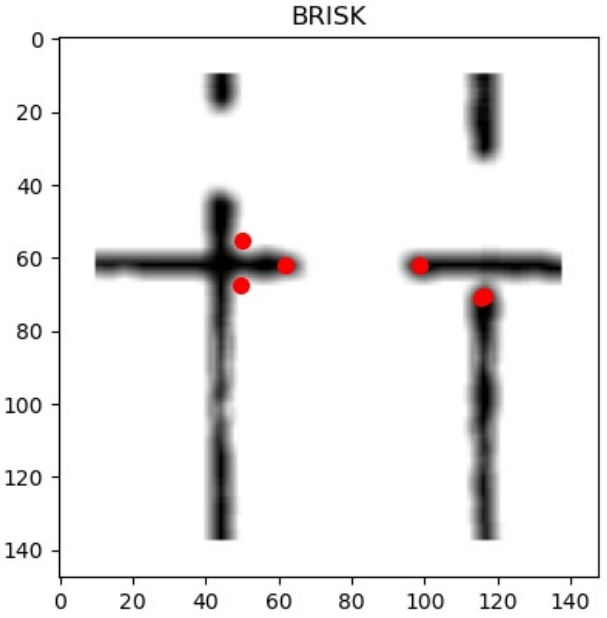






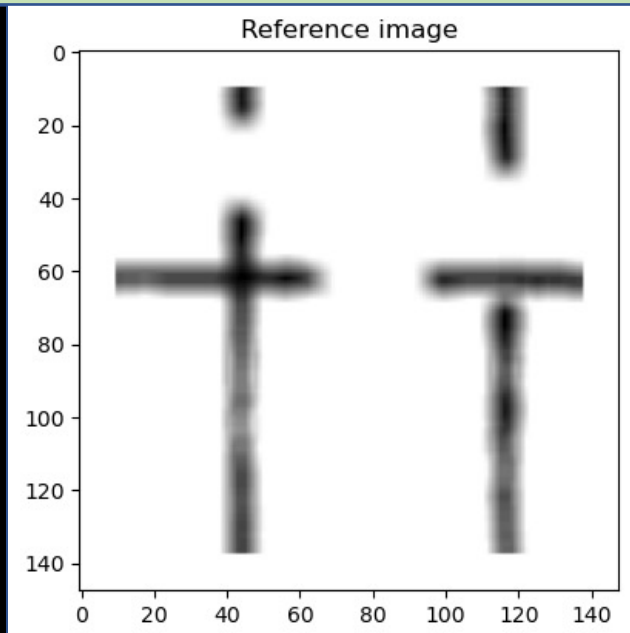


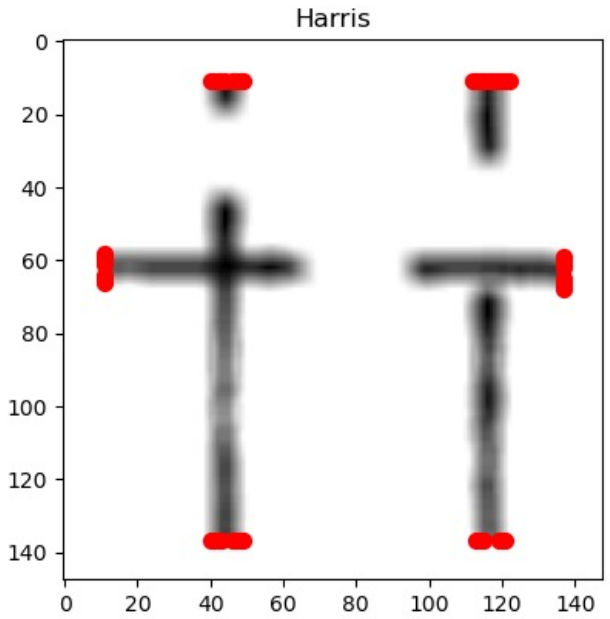


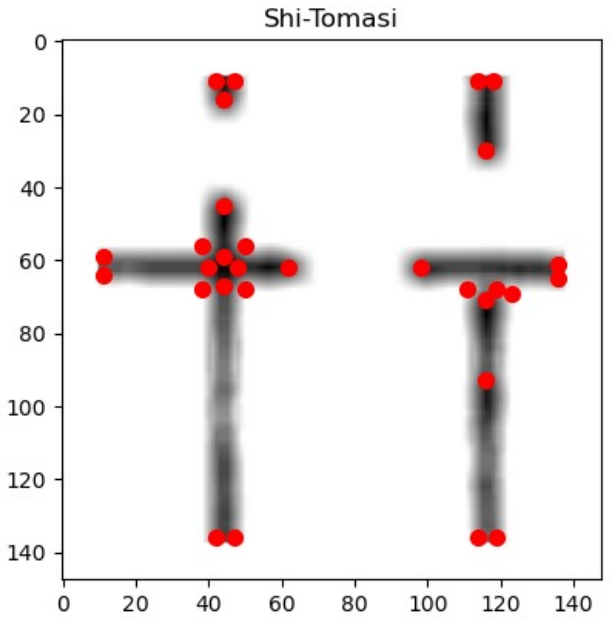


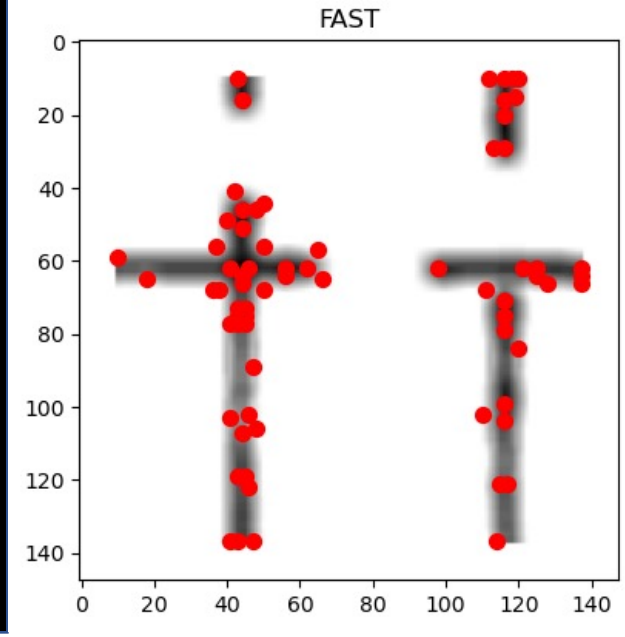
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

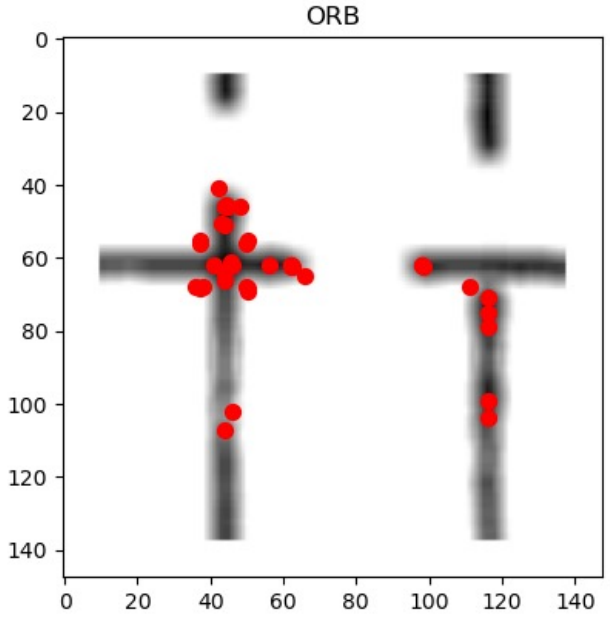
Gaussian blur (kernel 7 by 7)

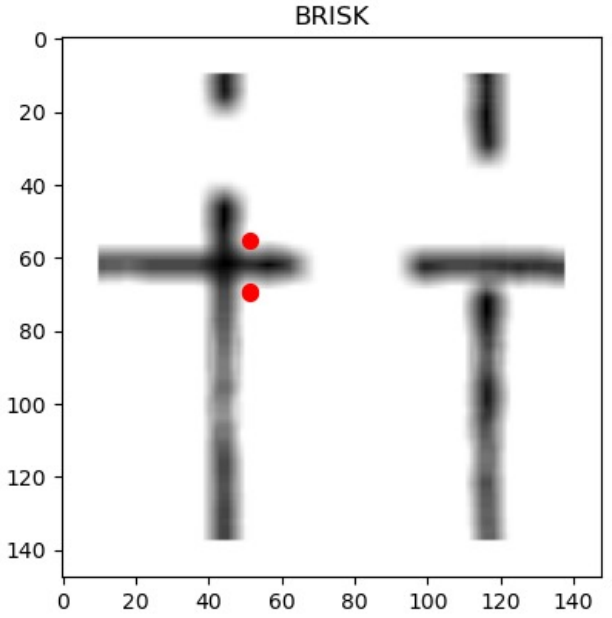






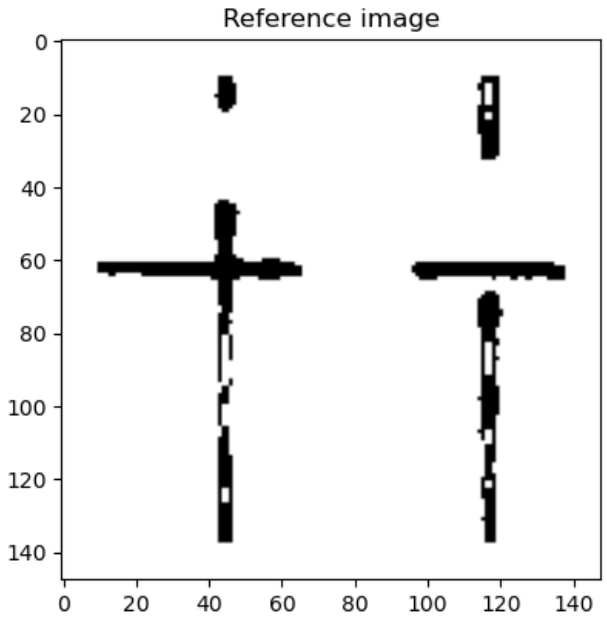


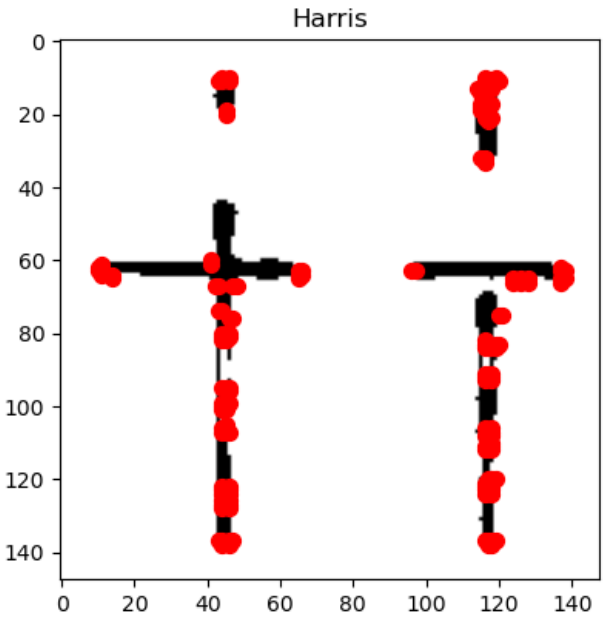


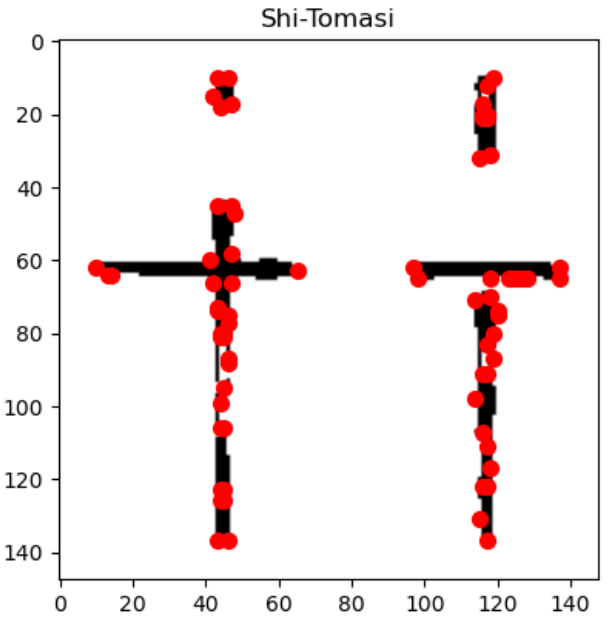


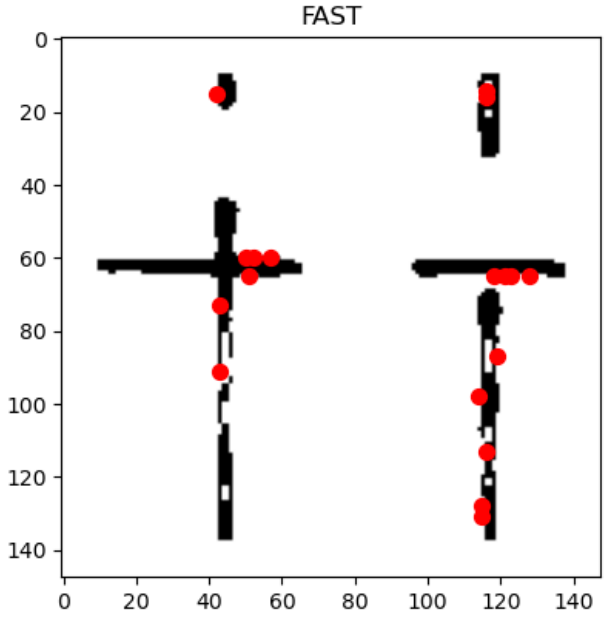
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

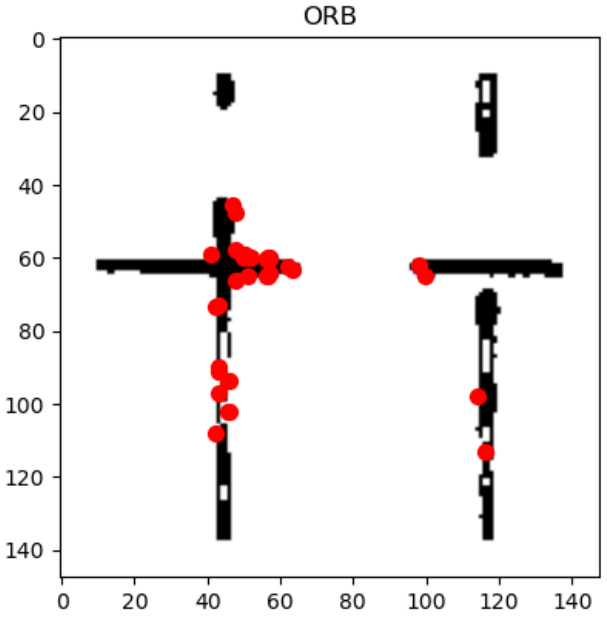
Dilate
(kernel 2 by 2)

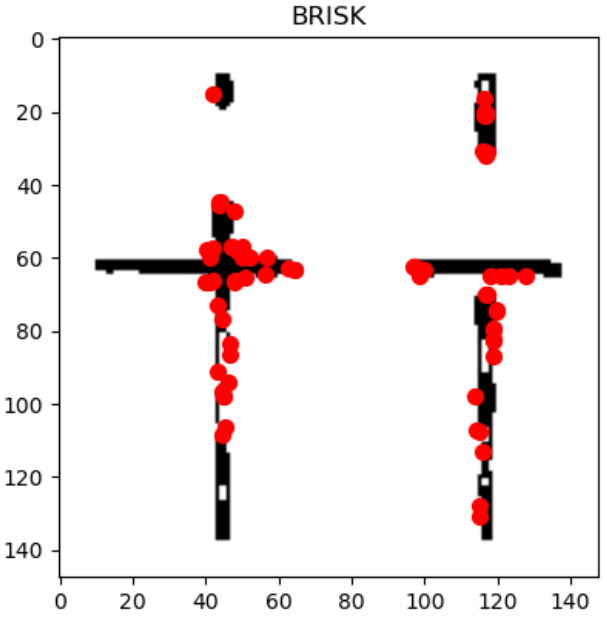






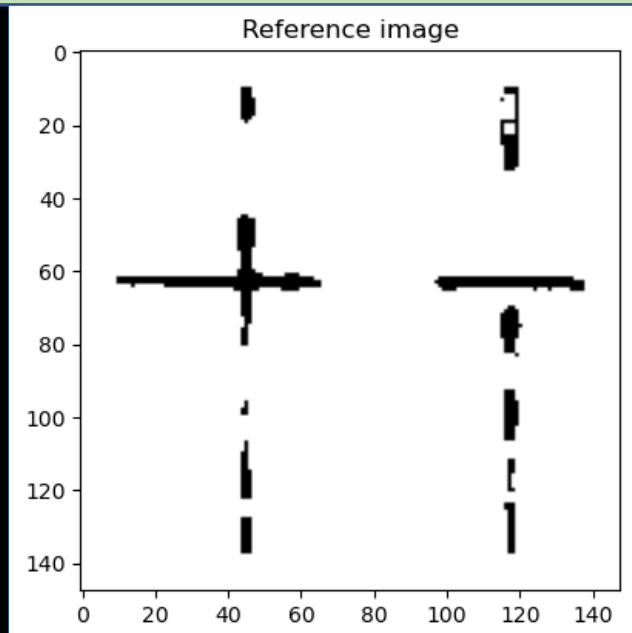


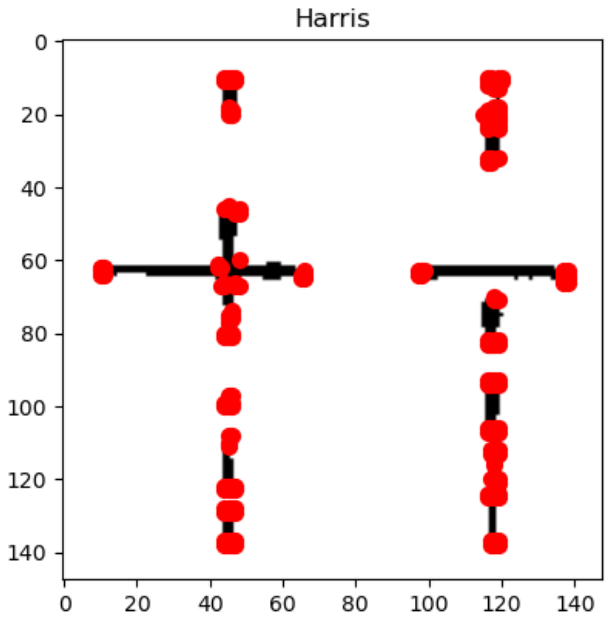


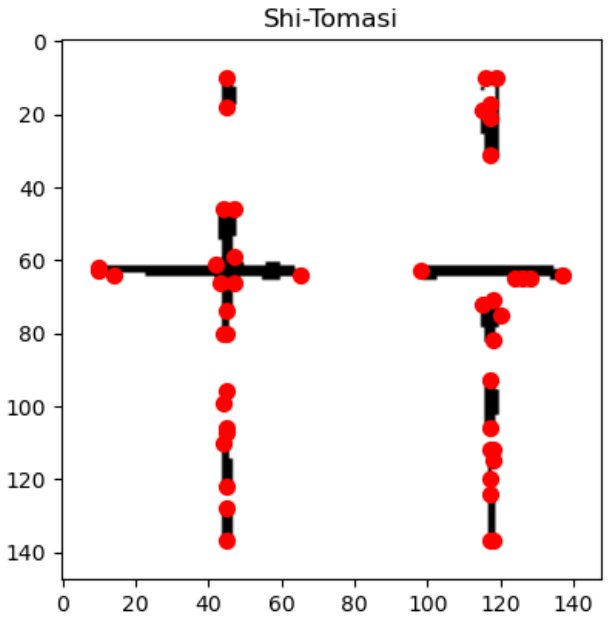


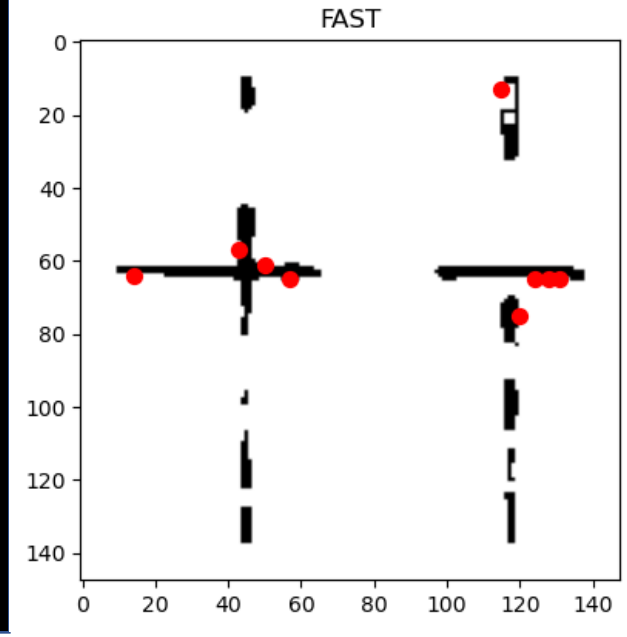
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

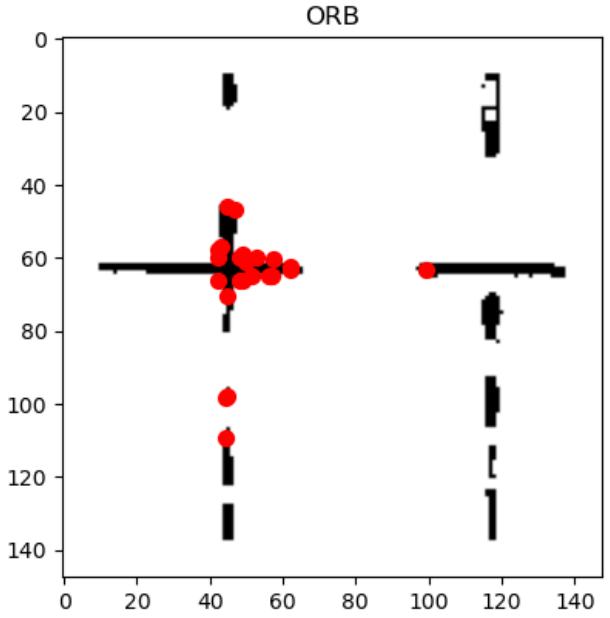
Dilate x2
(kernel 2 by 2)

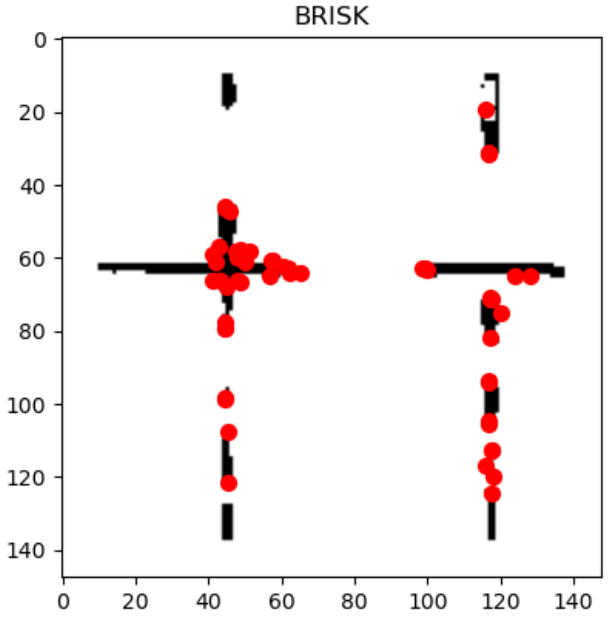






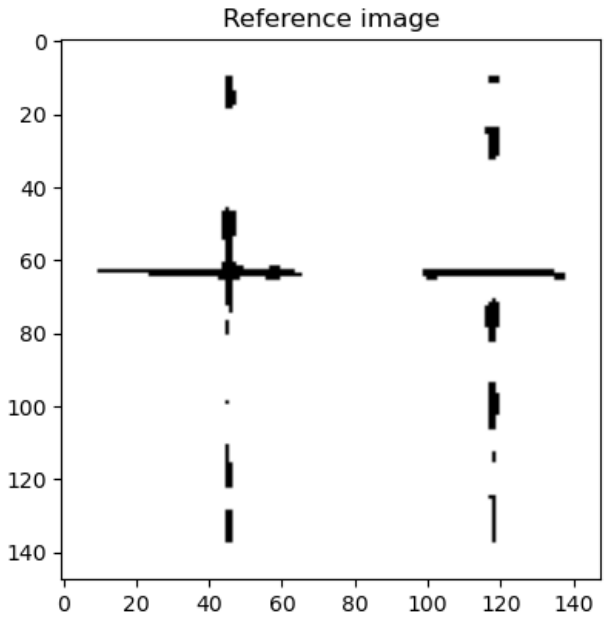


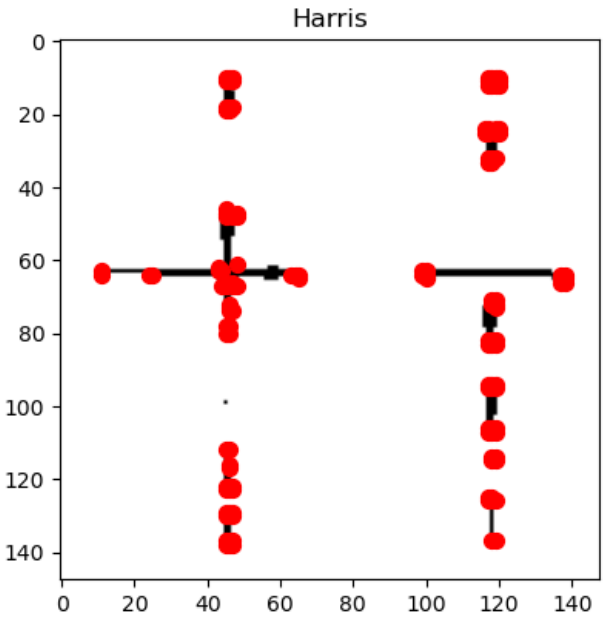


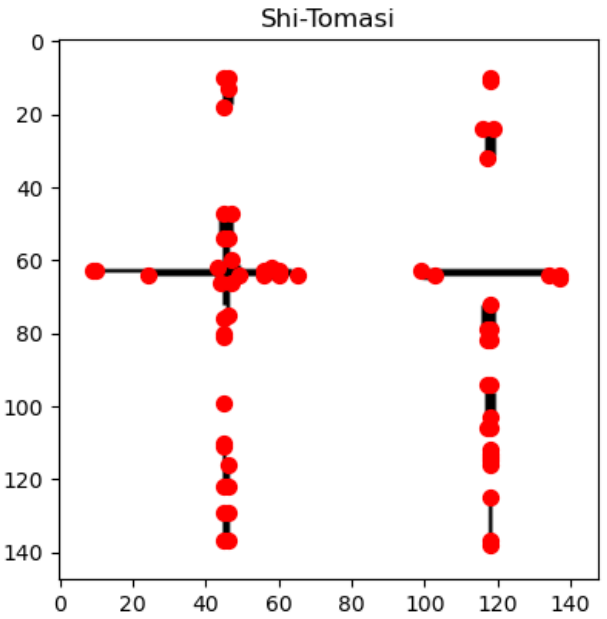


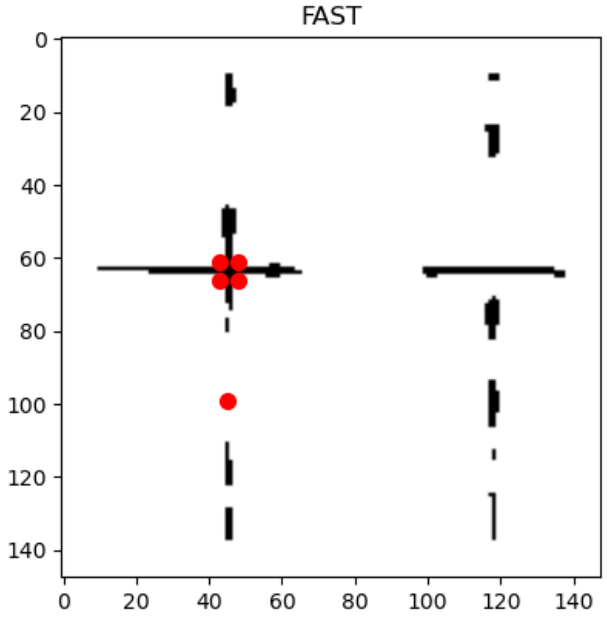
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

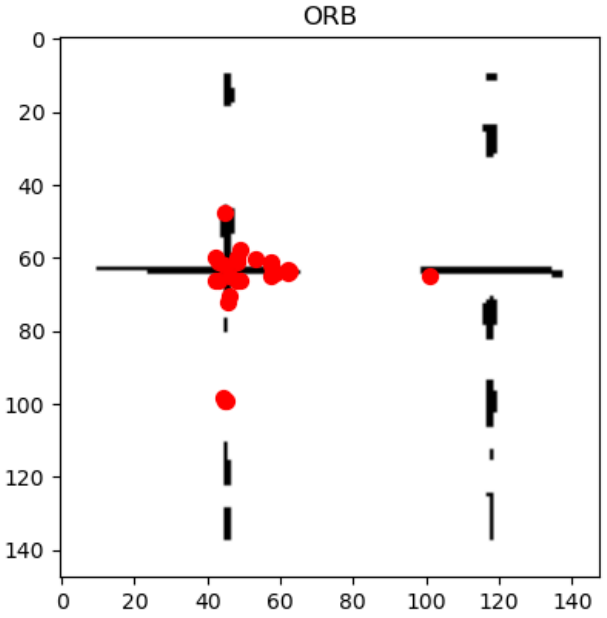
Dilate x3
(kernel 2 by 2)

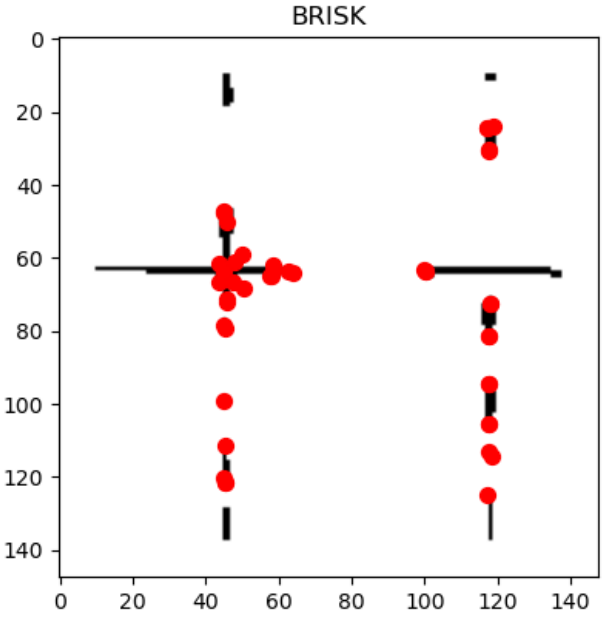






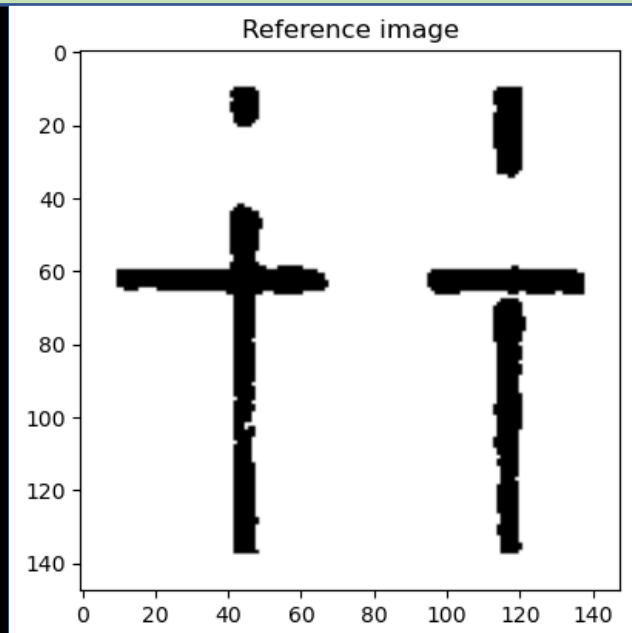


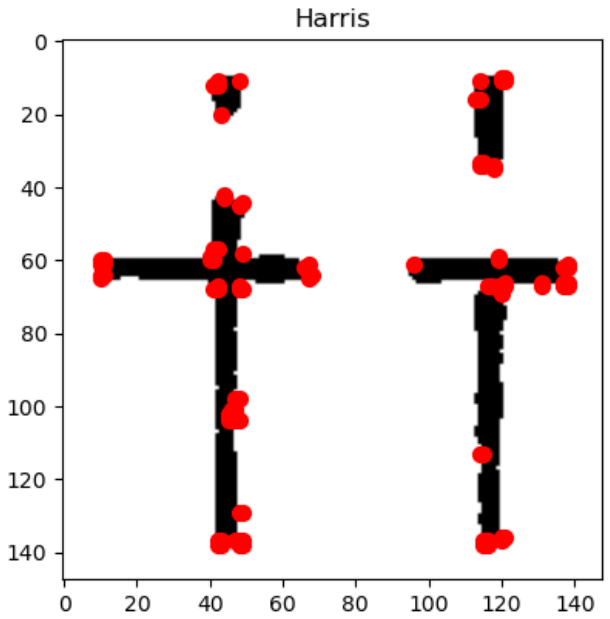


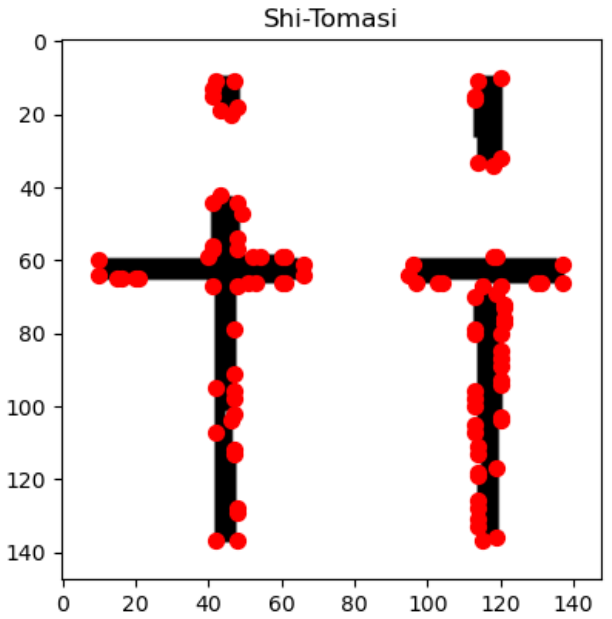


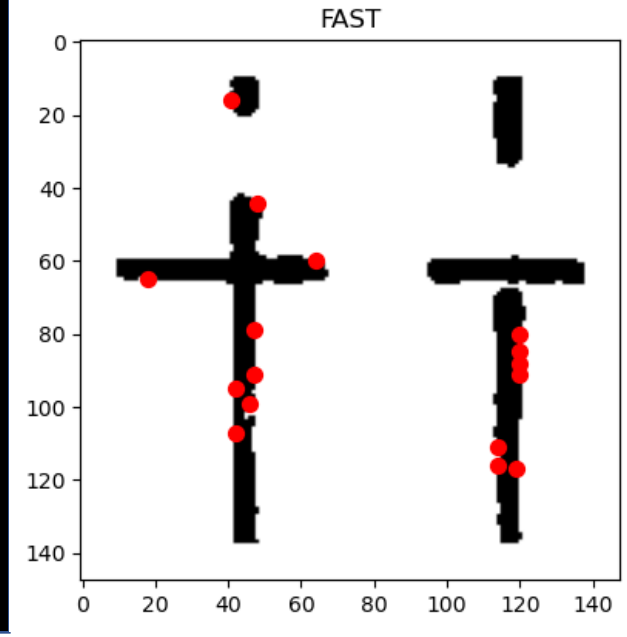
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

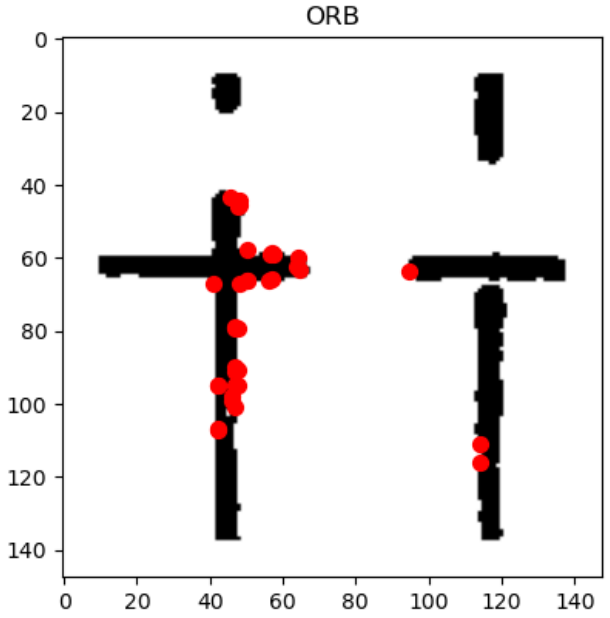
Erode
(kernel 2 by 2)

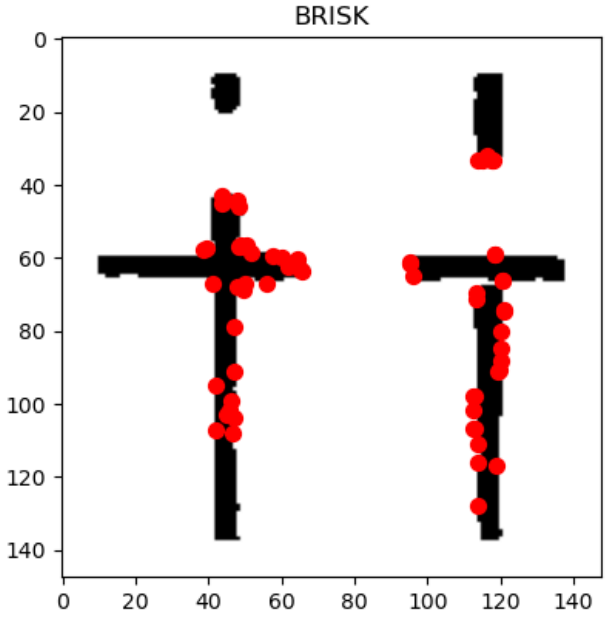






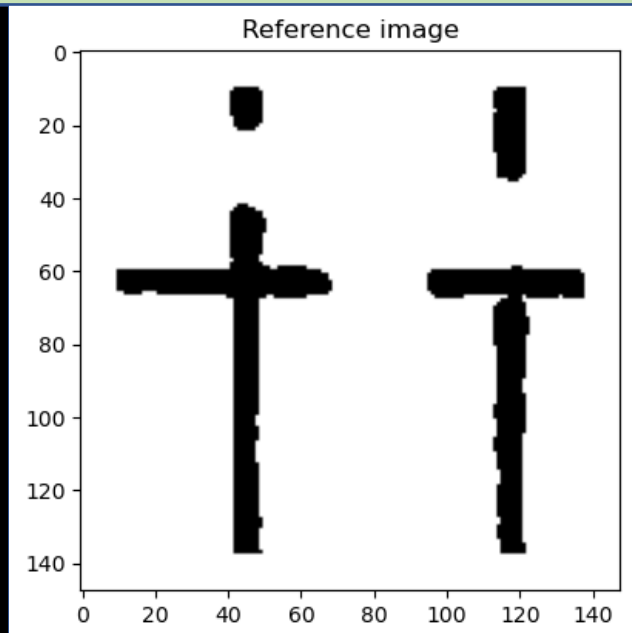


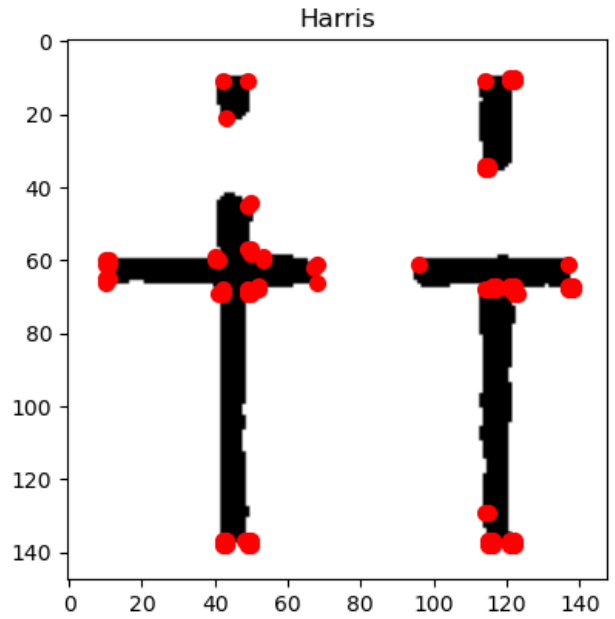


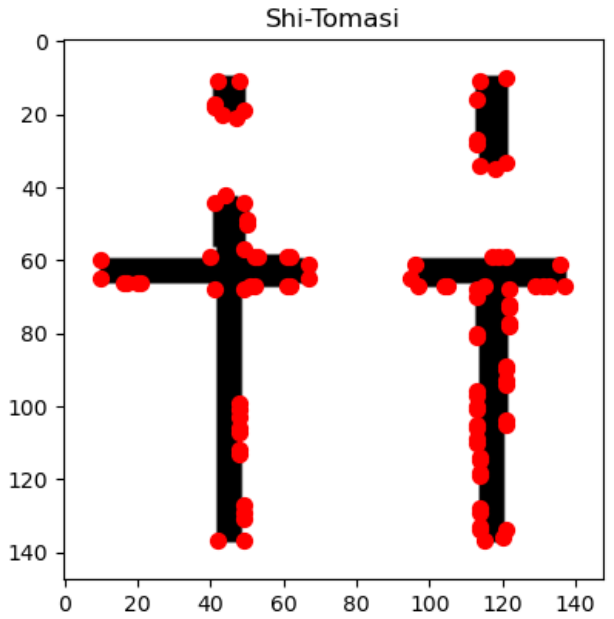


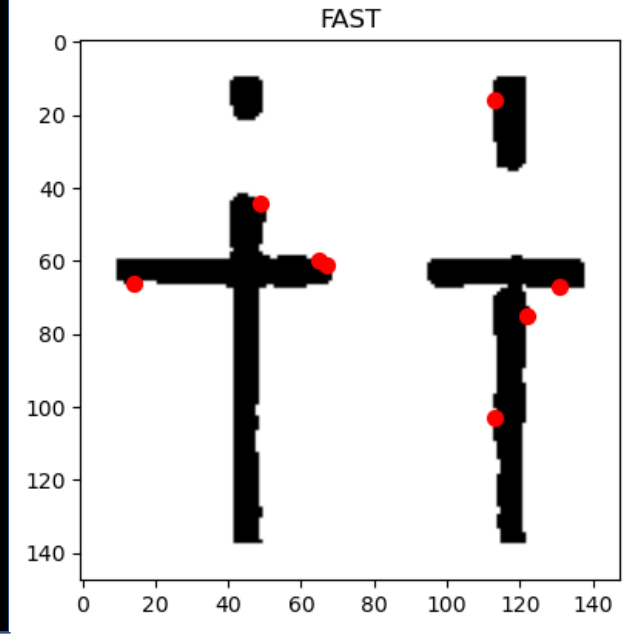
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

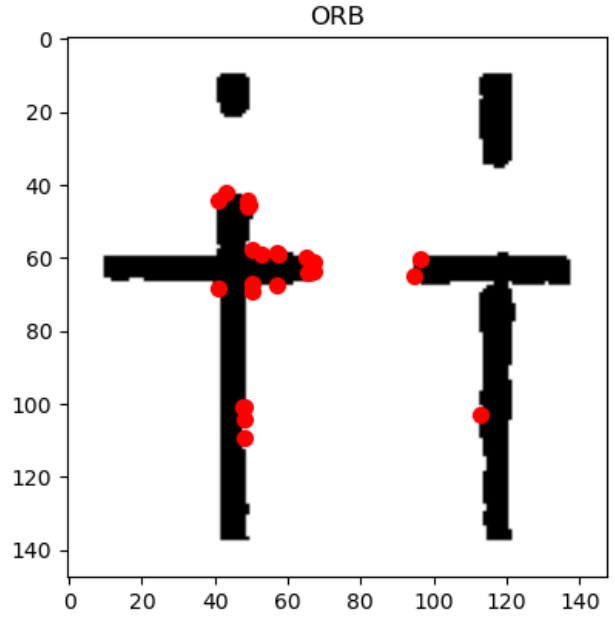
Erode x2
(kernel 2 by 2)

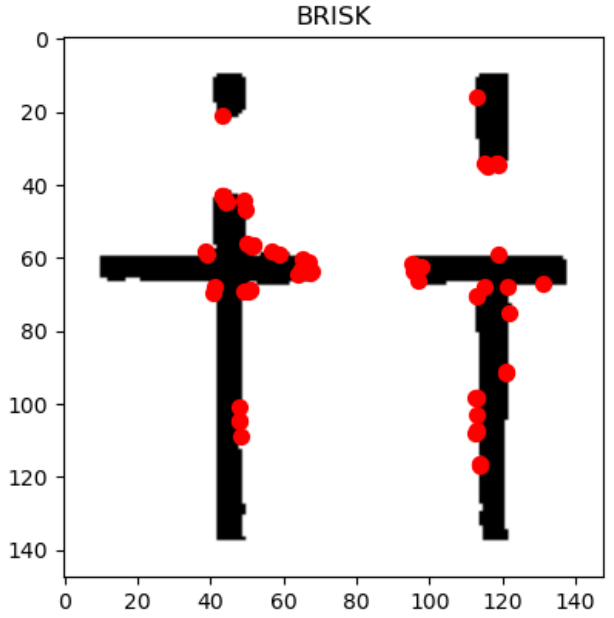






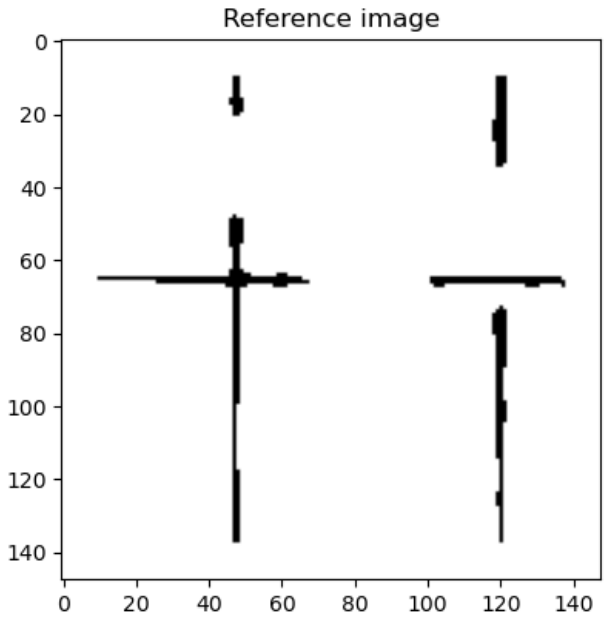


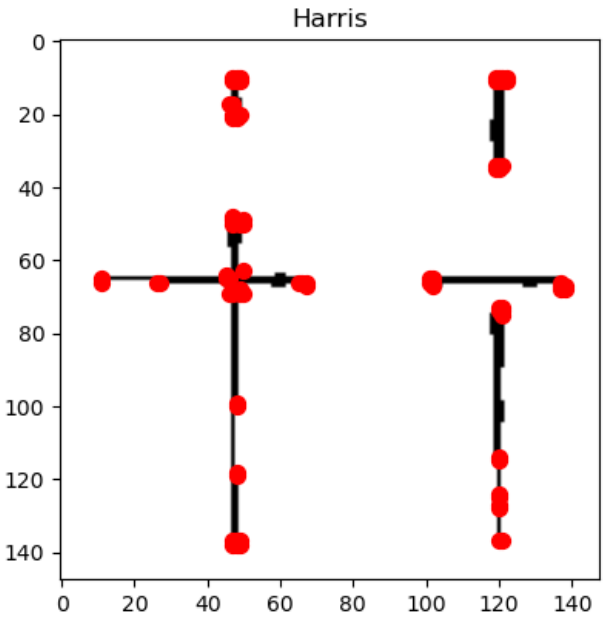


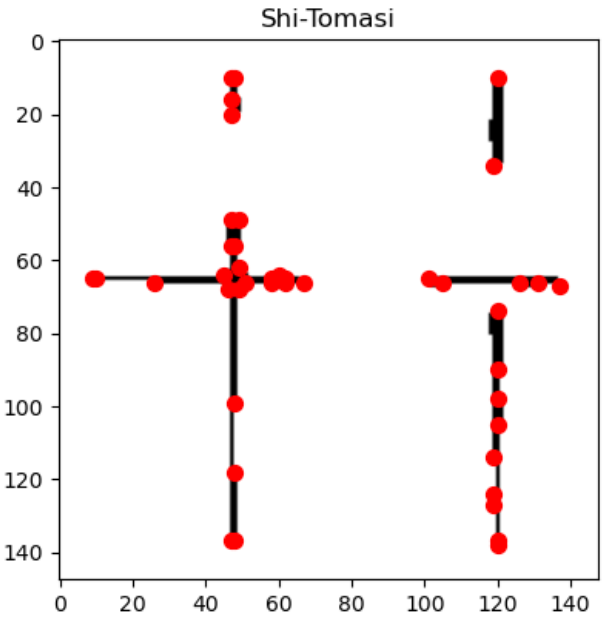


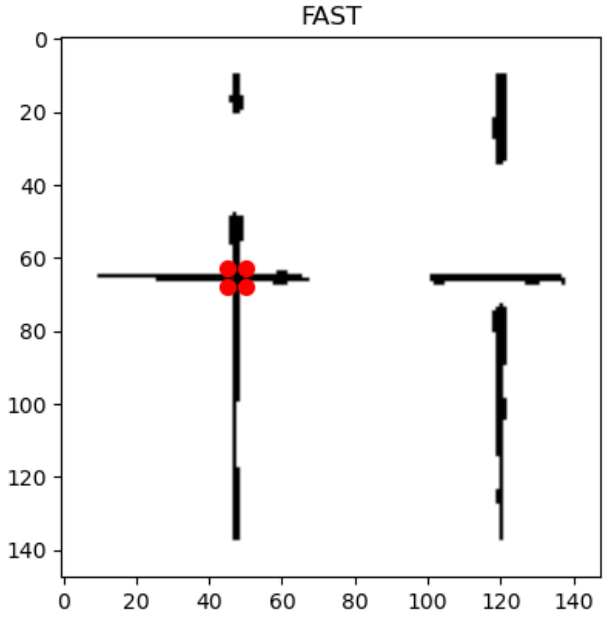
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

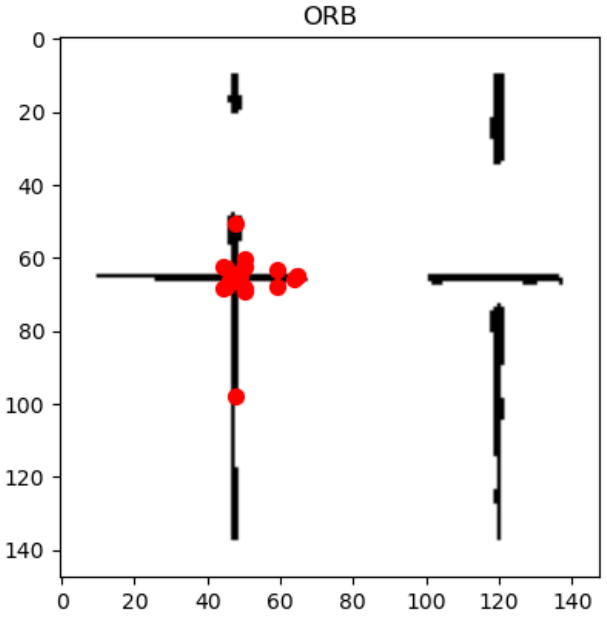
Erode x2
Dilate x5
(kernel 2 by 2)

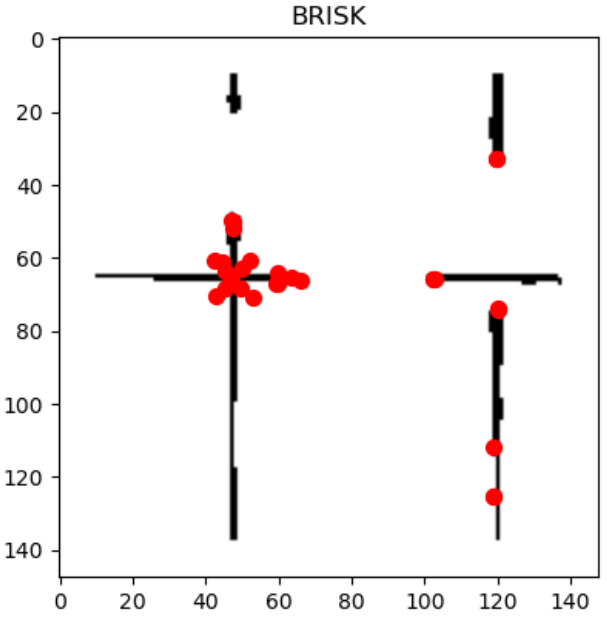






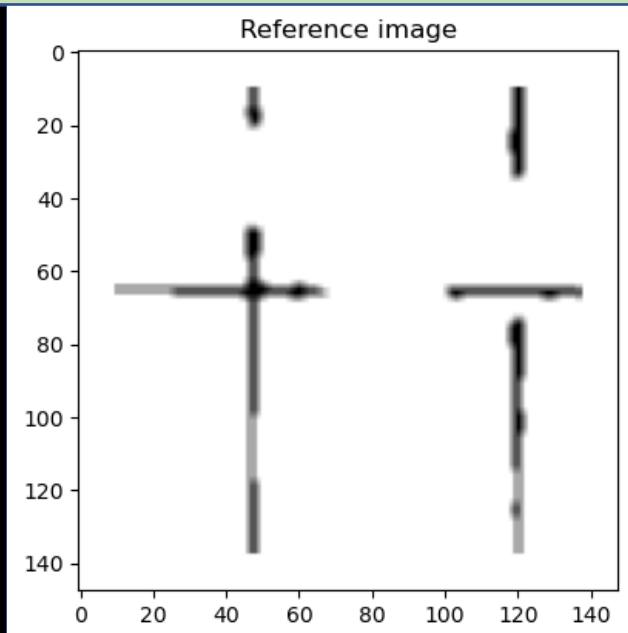


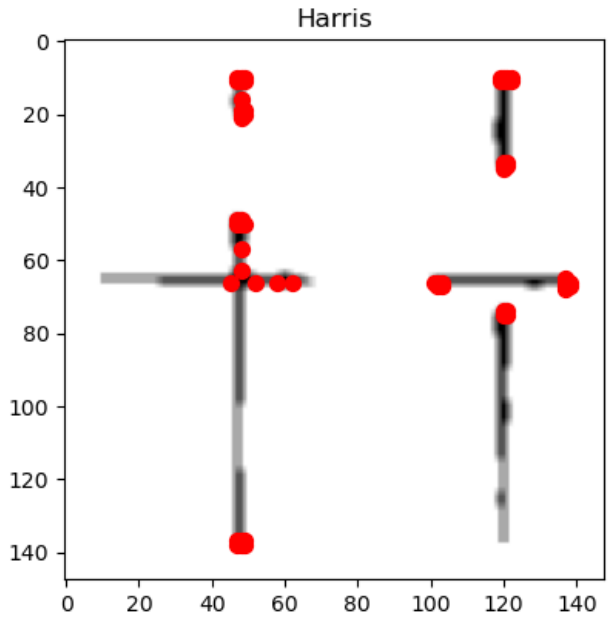


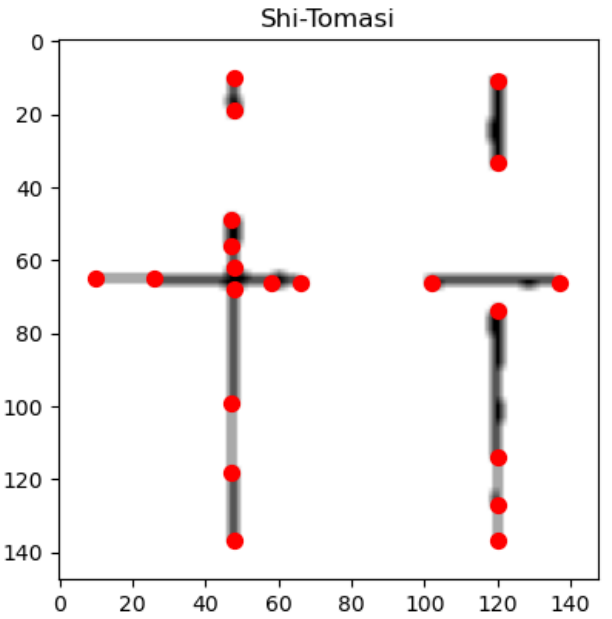


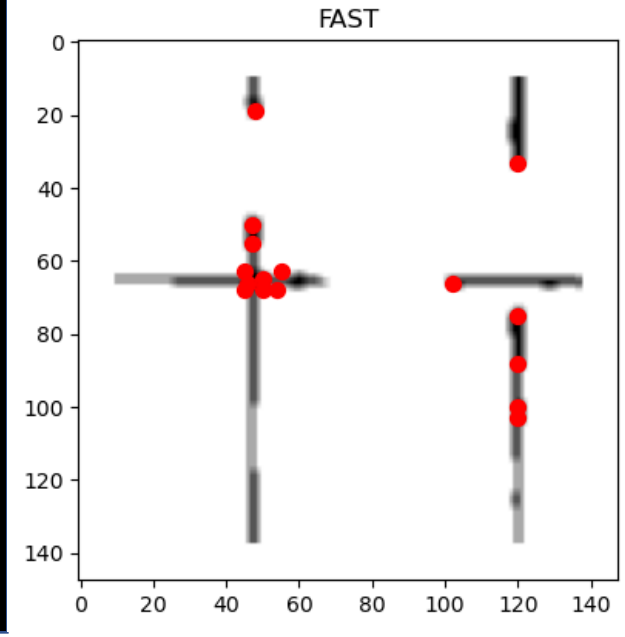
Part 1: Detecting corners from actual image using **Shi-Tomasi algorithm**

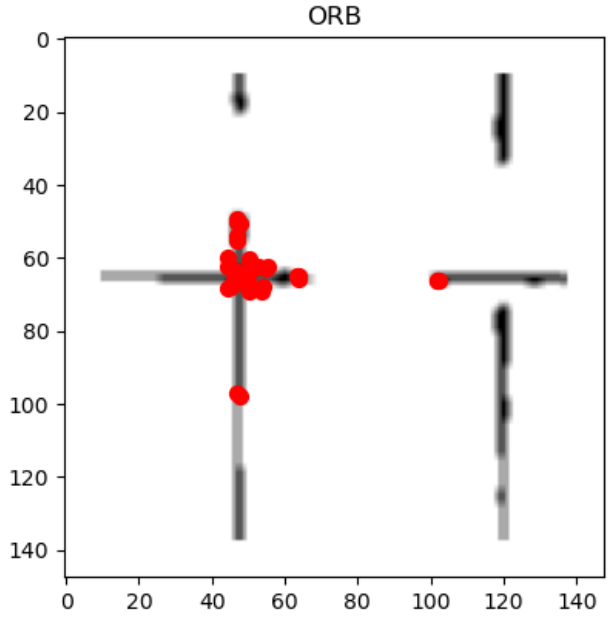
Erode x2
Dilate x5
(kernel 2 by 2)
Gaussian blur
(kernel 3 by 3)

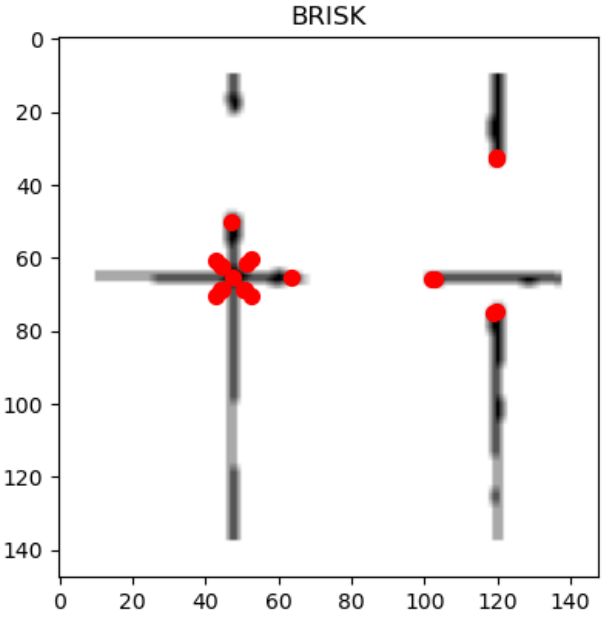




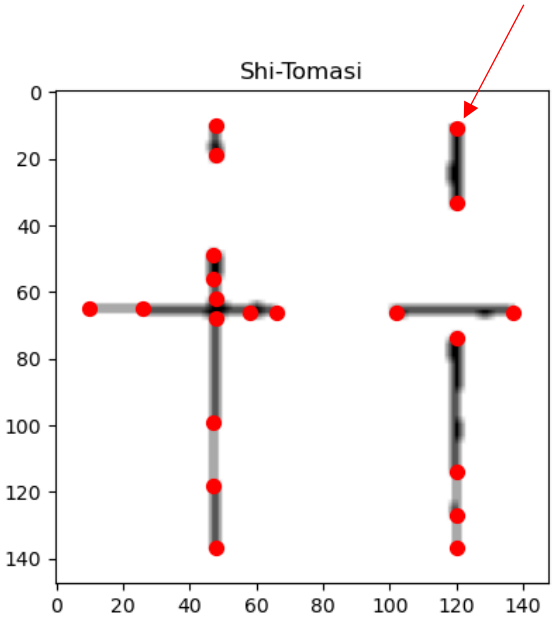


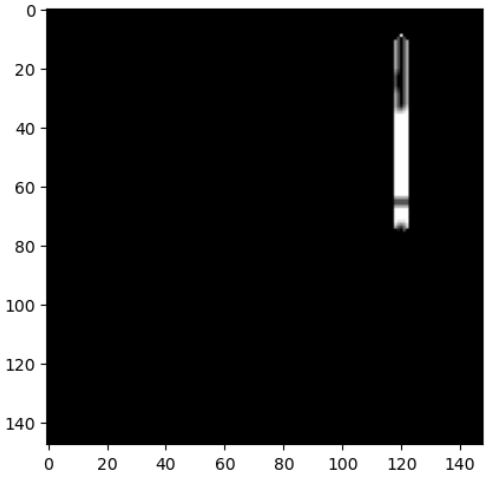




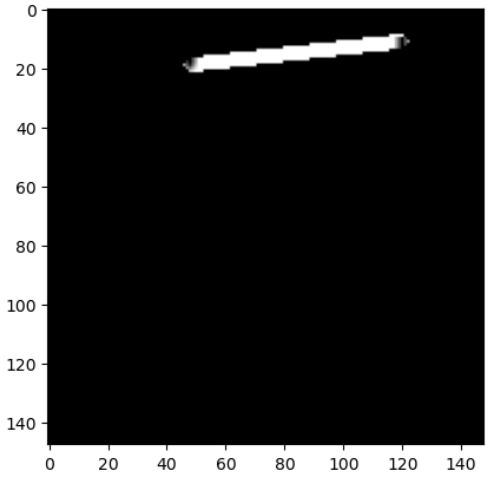


Part 2: Line coordinate tracing

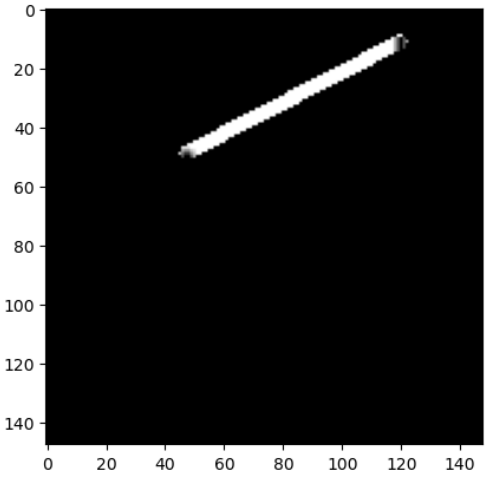




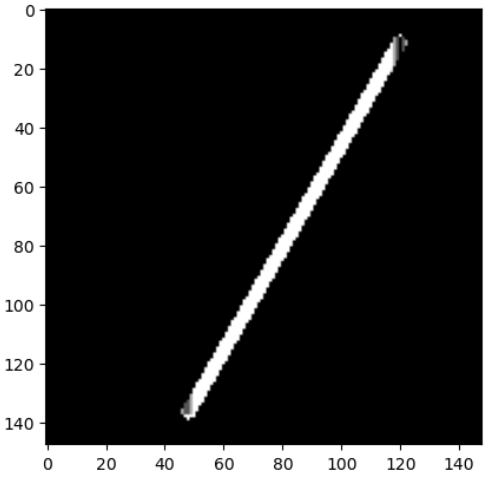
Mean = 2.1300



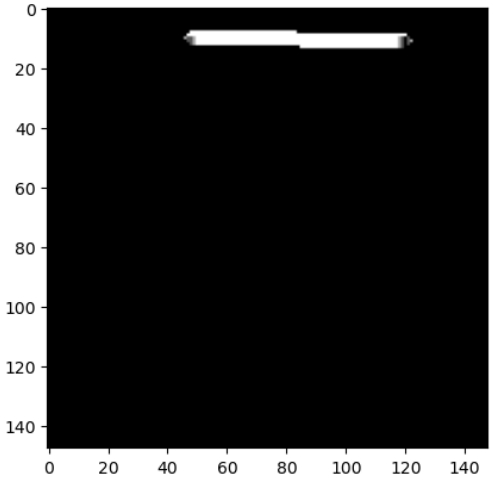
2.8426



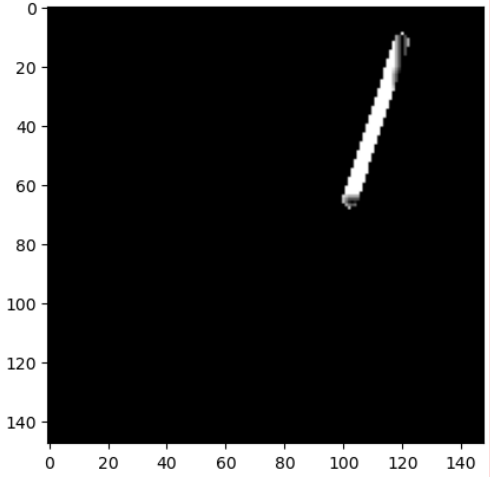
2.8400



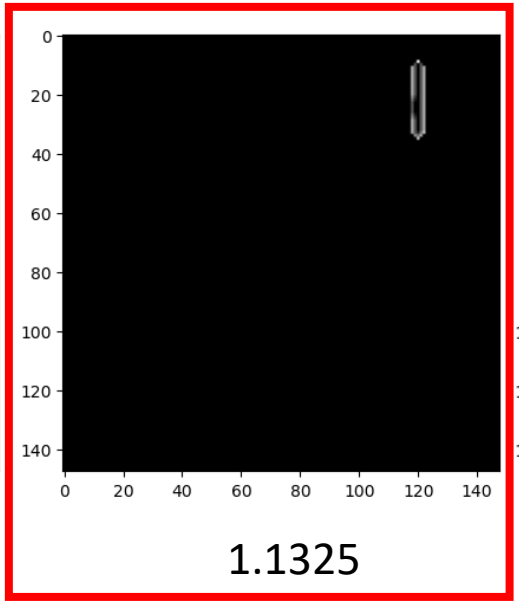
2.8837



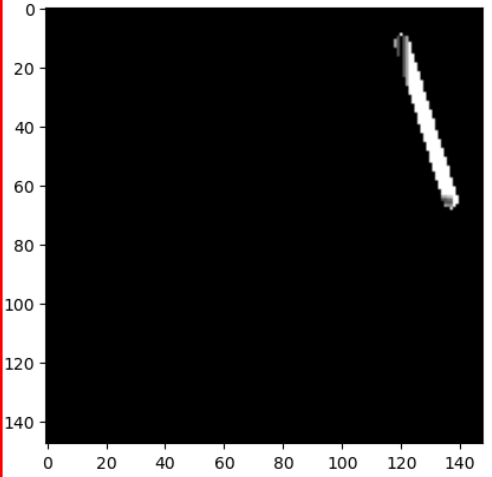
2.8790



2.5800

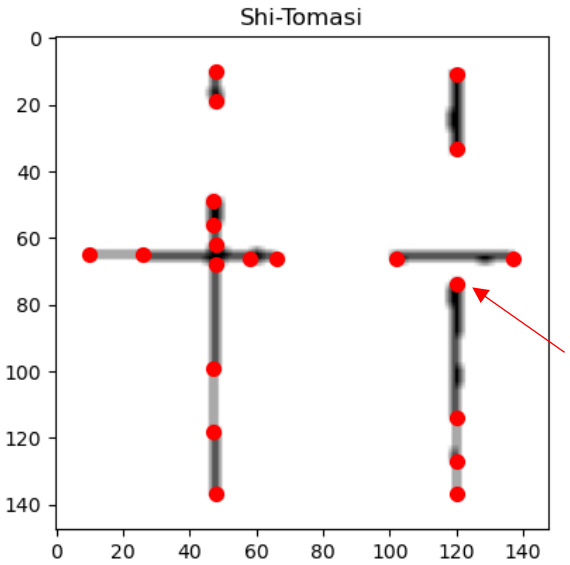


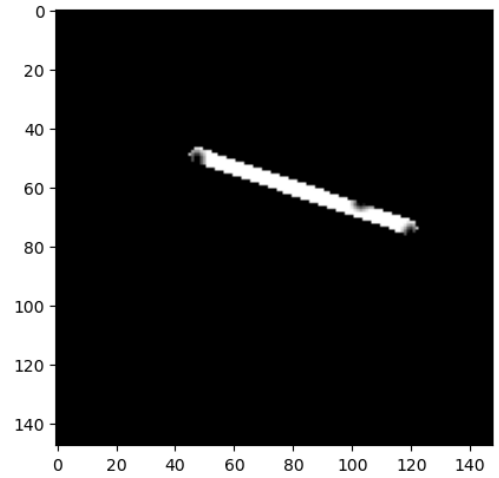
1.1325



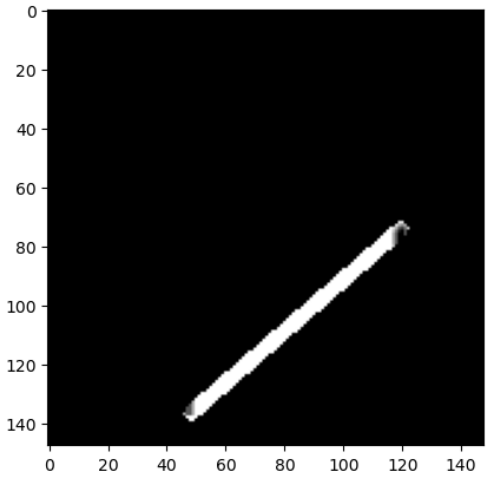
2.6412

Part 2: Line coordinate tracing

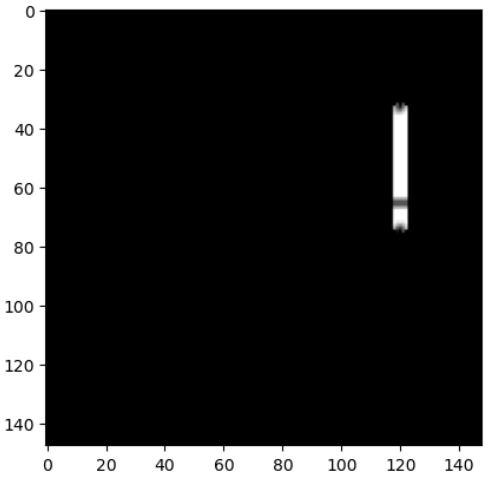




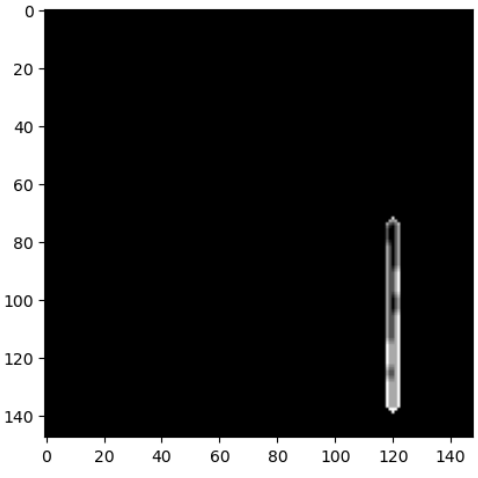
Mean = 2.7615



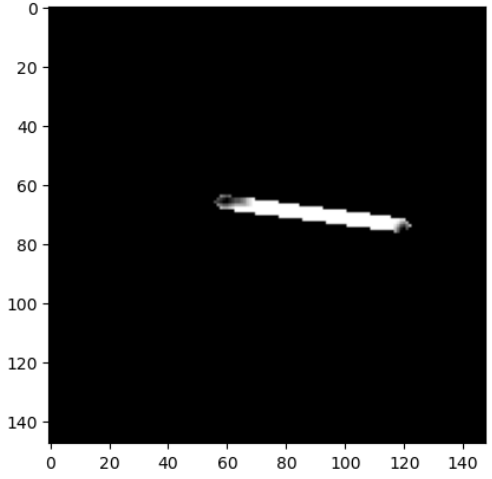
2.8636



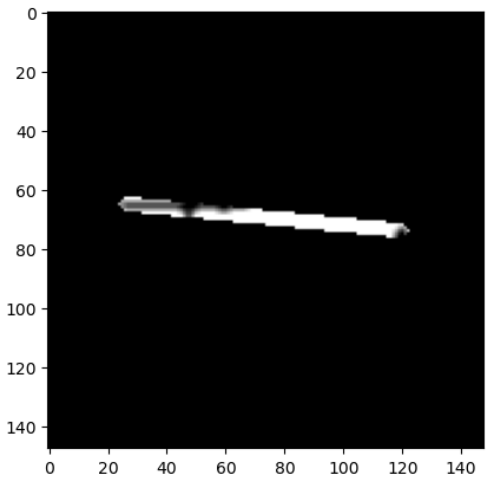
2.6405



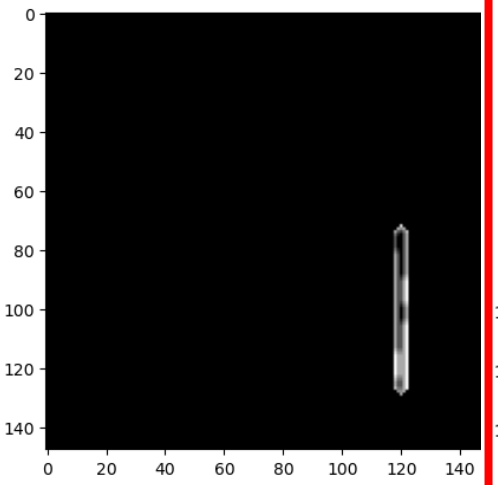
1.7594



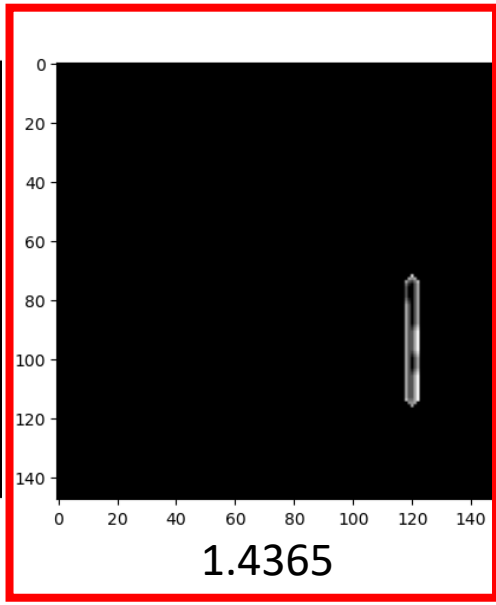
2.6686



2.4418

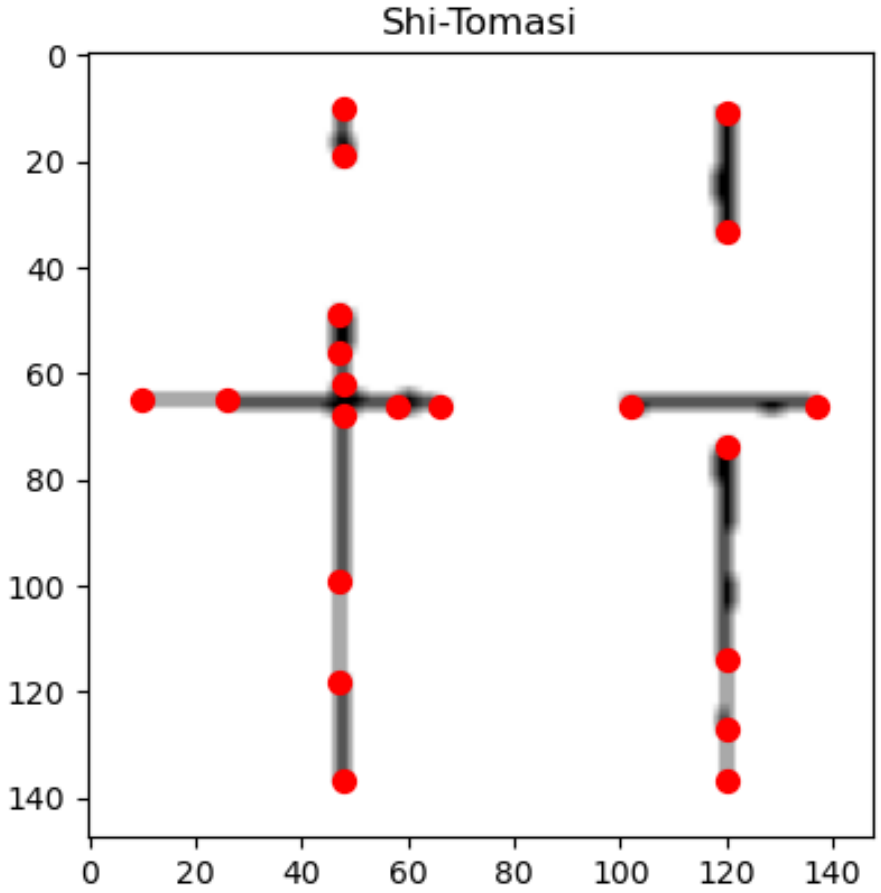


1.6313



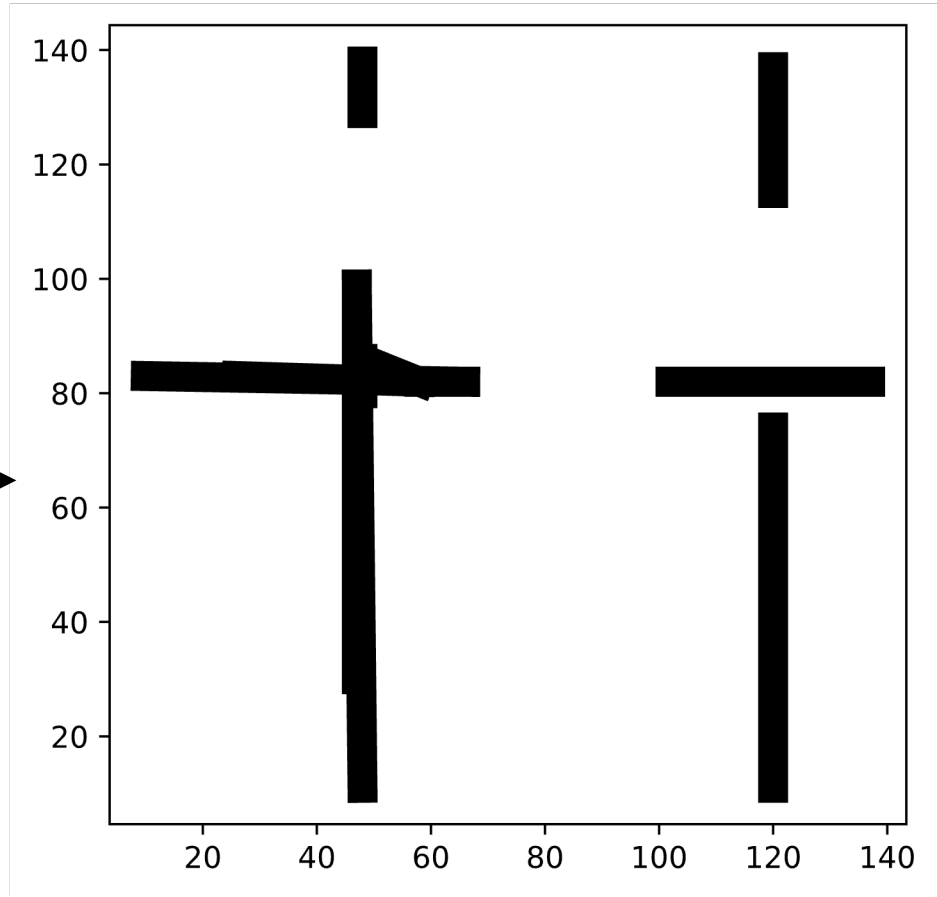
1.4365

Part 2: Line coordinate tracing



List of coordinate pairs:

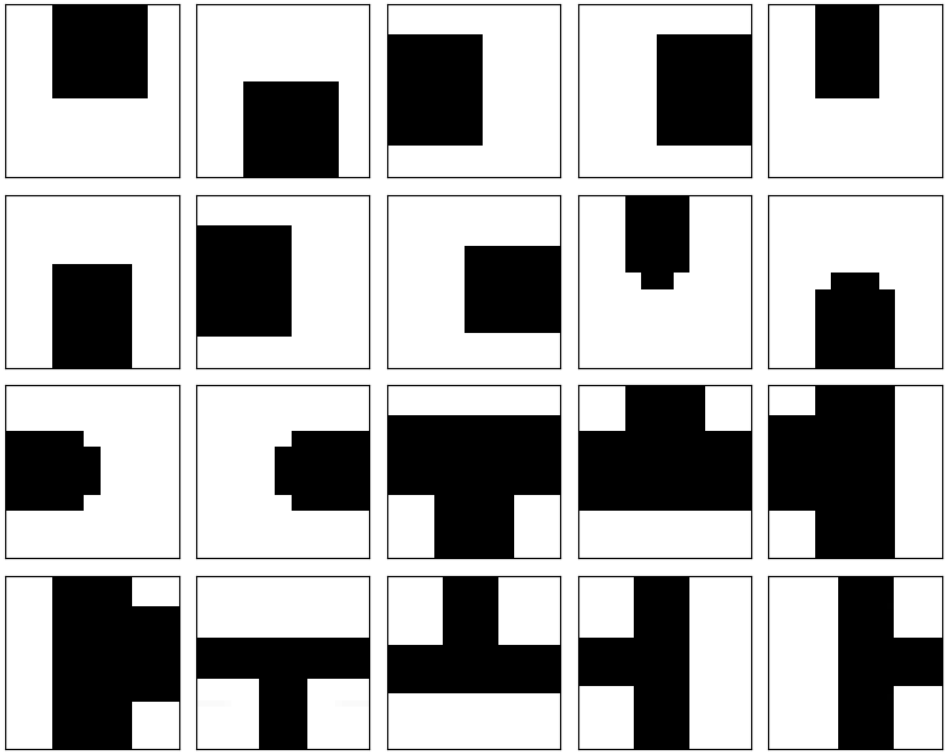
((120, 137), (120, 115)),
((120, 74), (120, 34)),
((48, 129), (48, 138)),
((47, 99), (47, 92)),
((48, 11), (47, 99)),
....



Part 1: Detecting corners from actual image using **template matching**

Erode x1
Dilate x2
(kernel 2 by 2)
Gaussian blur
(kernel 3 by 3)

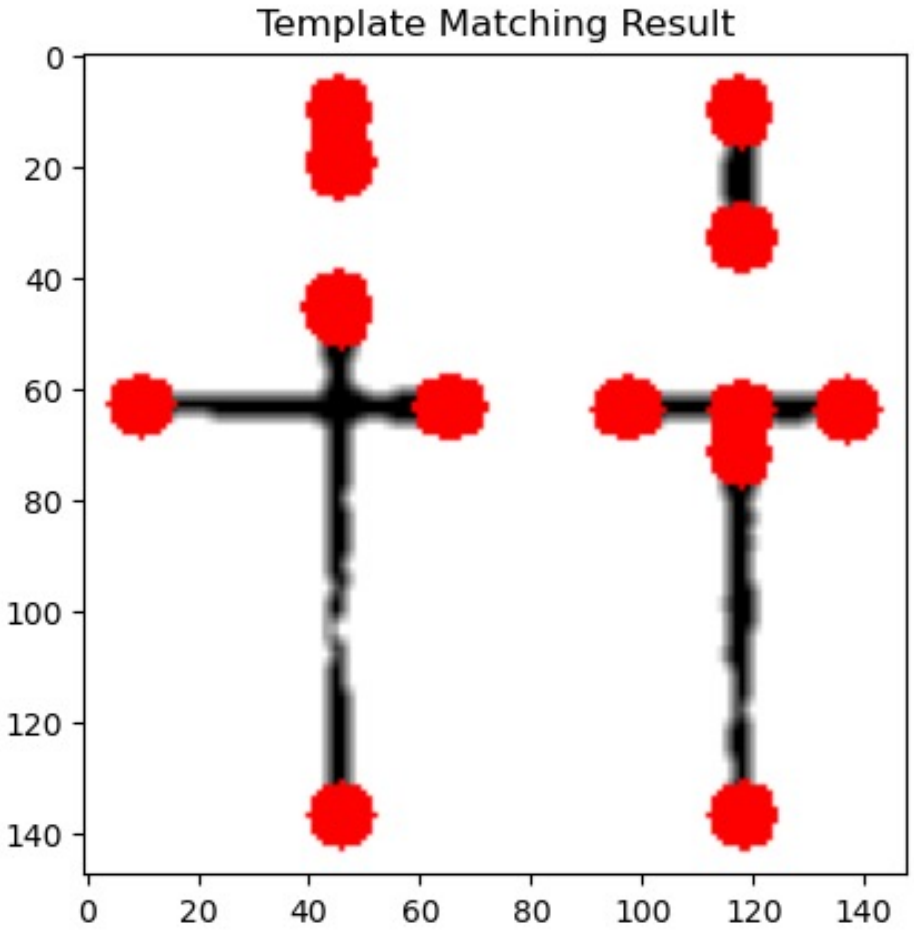
Filters/templates:

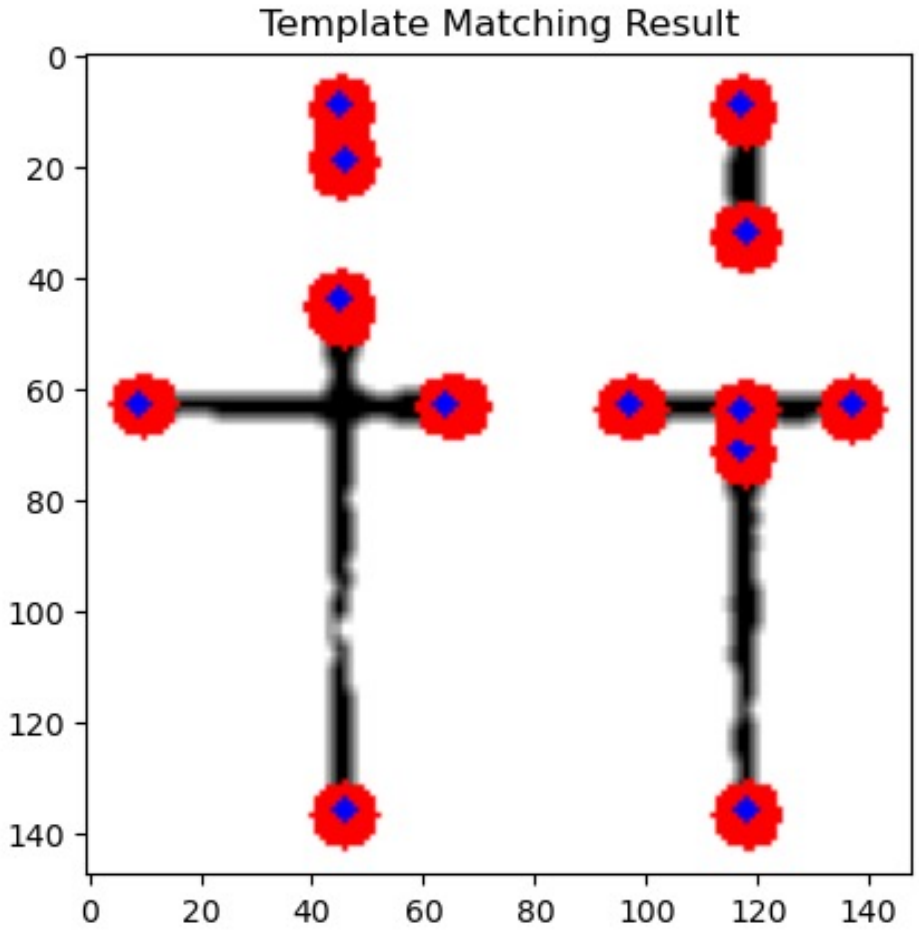


- 11 by 11 pixels
- 25 by 25 pixels (last 4)
- Normalized cross-correlation

Part 1: Detecting corners from actual image using **template matching**

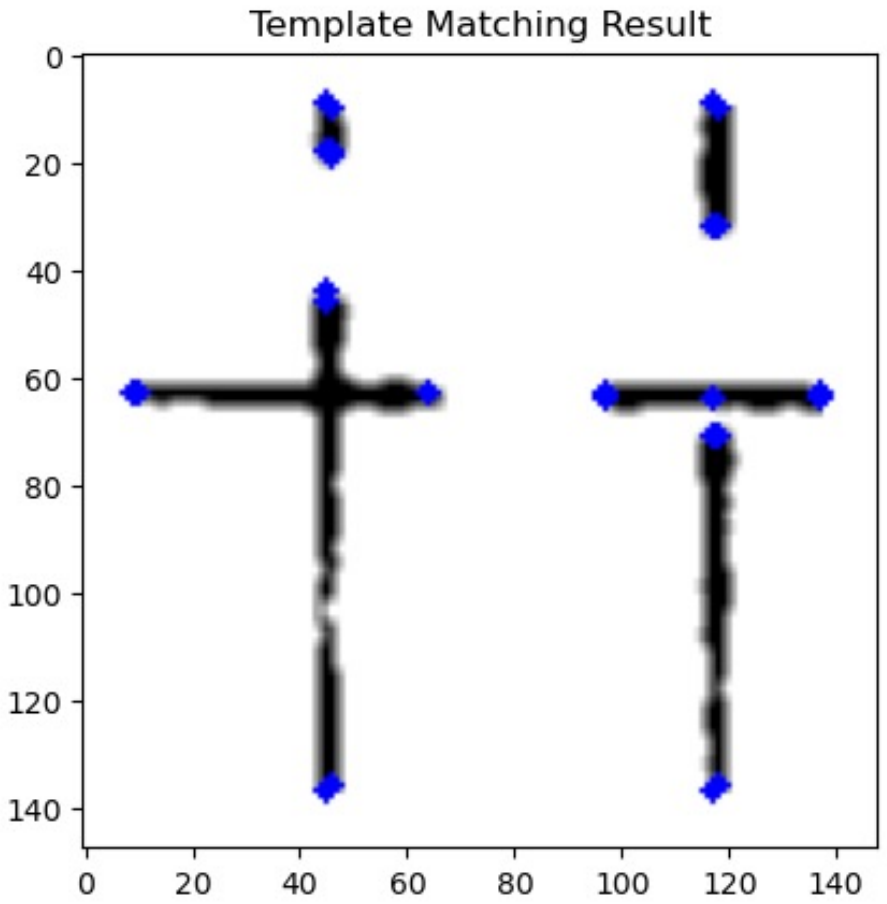
Erode x1
Dilate x2
(kernel 2 by 2)
Gaussian blur
(kernel 3 by 3)





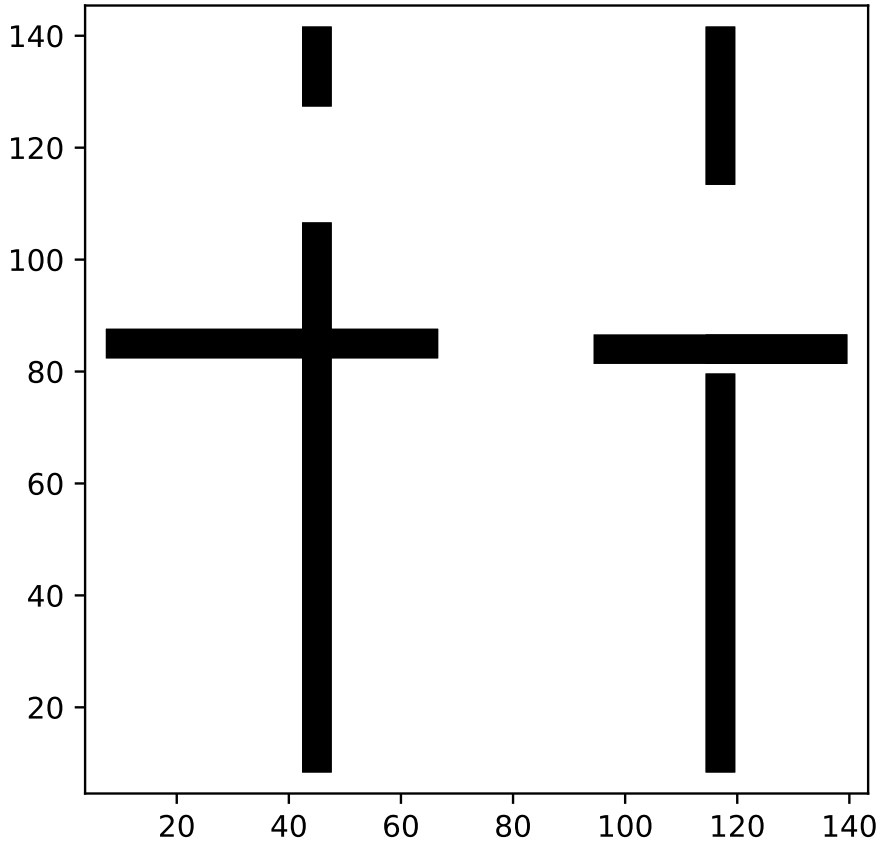
Filtering clustered coordinates into one representative (blue dots)

Part 2: Line coordinate tracing



List of coordinate pairs:

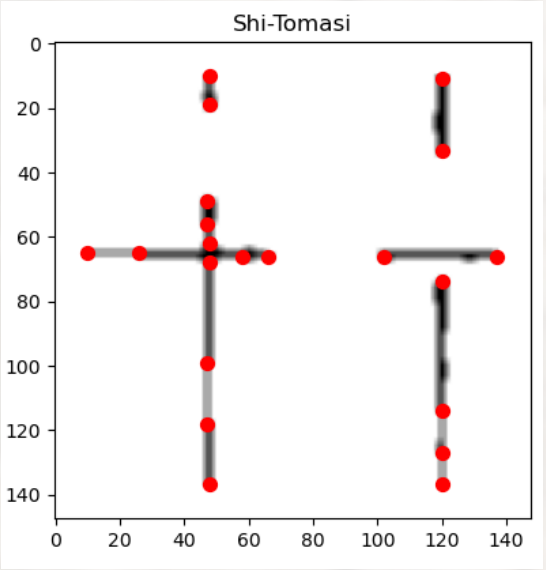
((45, 130), (45, 139)),
((117, 116), (117, 139)),
((45, 11), (45, 104)),
((117, 11), (117, 77)),
....

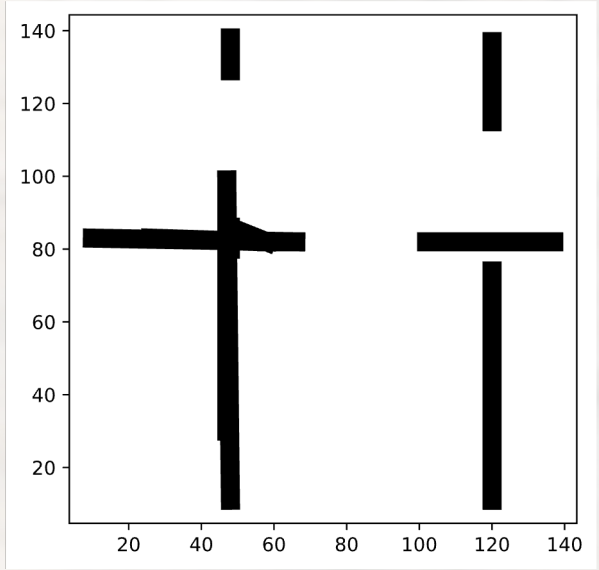


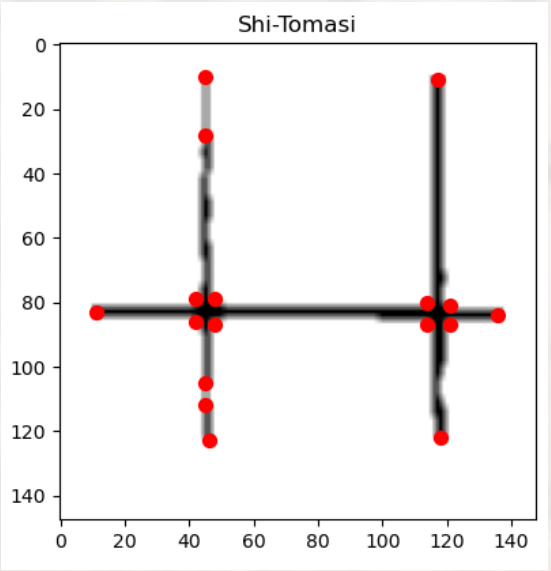
STAGE 2: VECTORIZATION

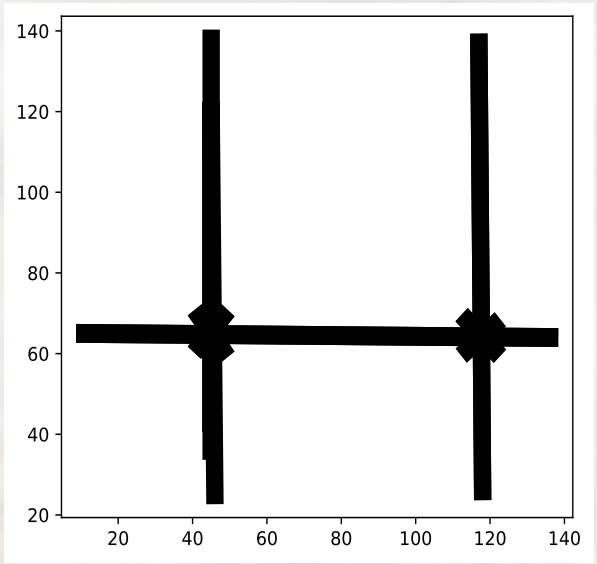
Comparison

SHI-TOMASI

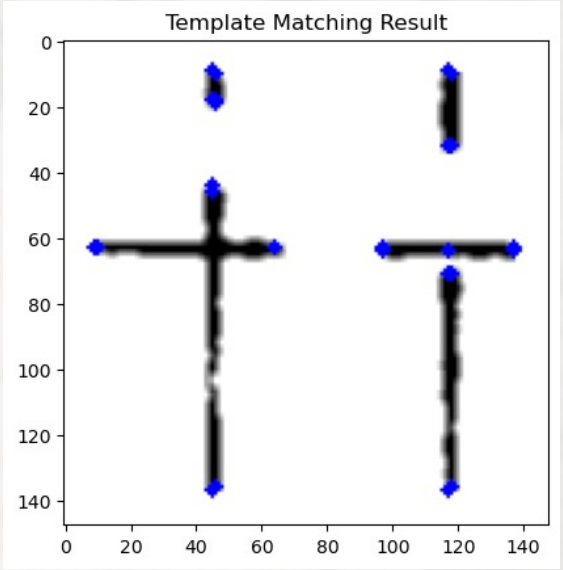


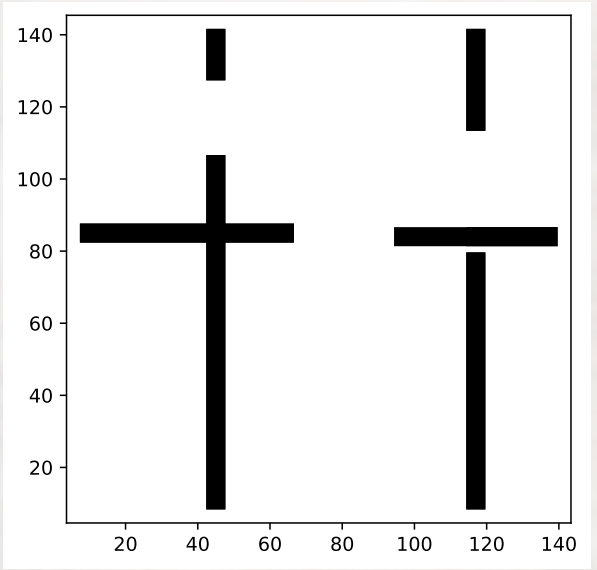


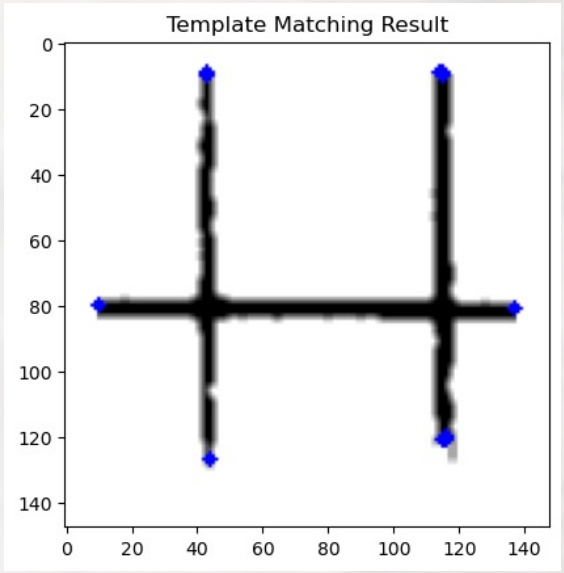


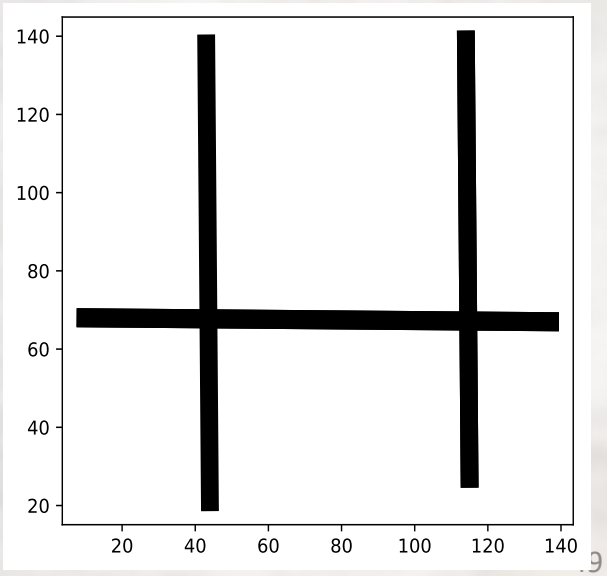


TEMPLATE MATCHING





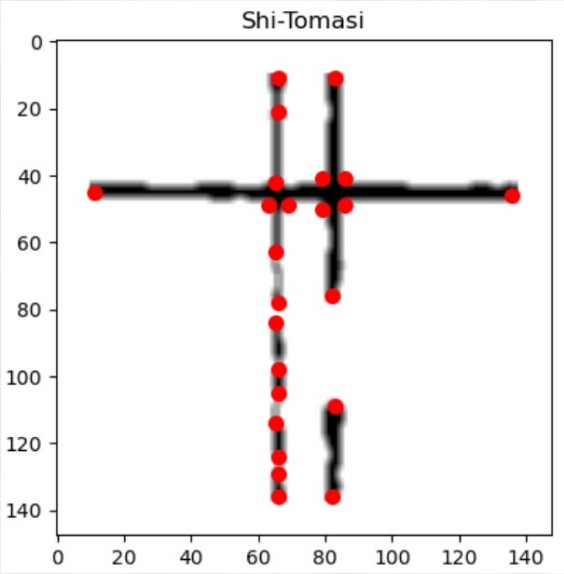


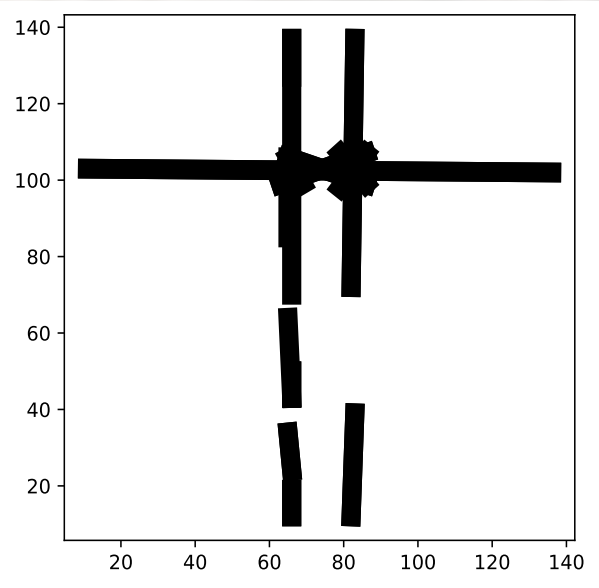


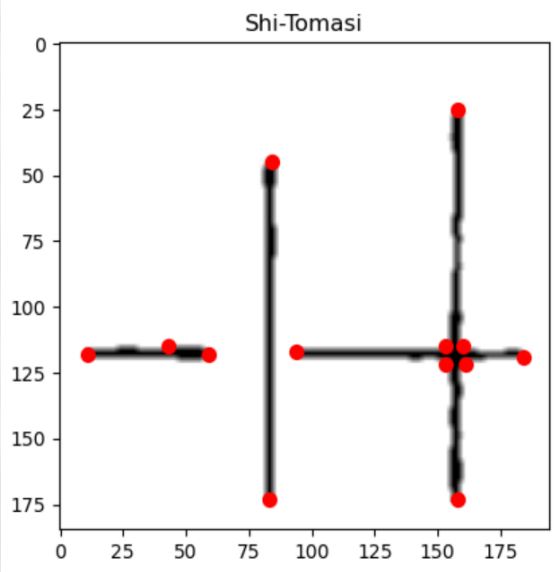
STAGE 2: VECTORIZATION

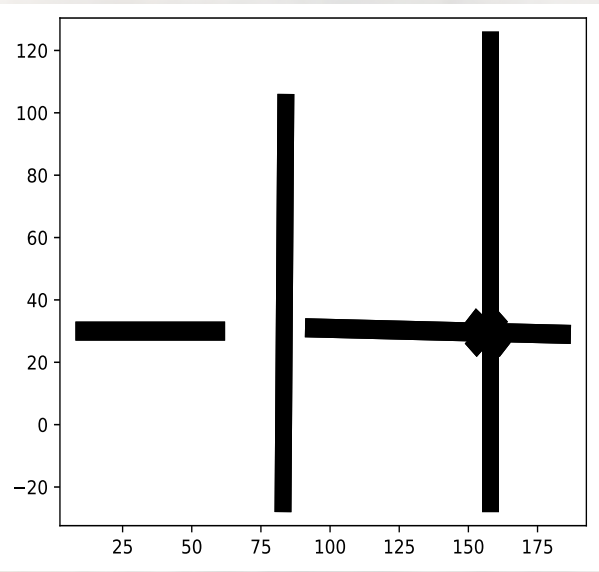
Comparison

SHI-TOMASI

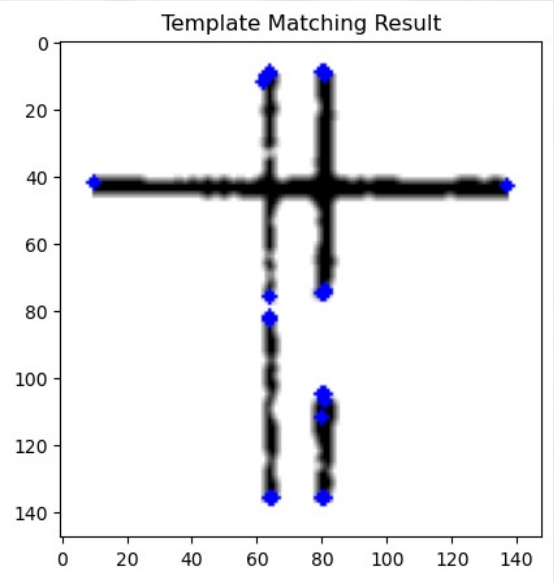


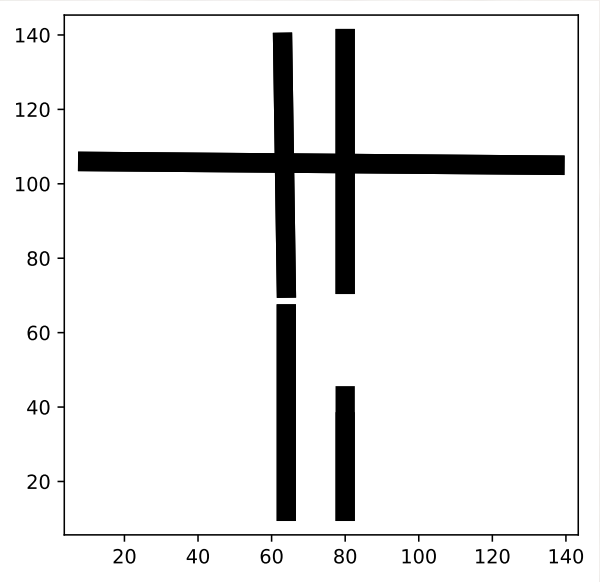


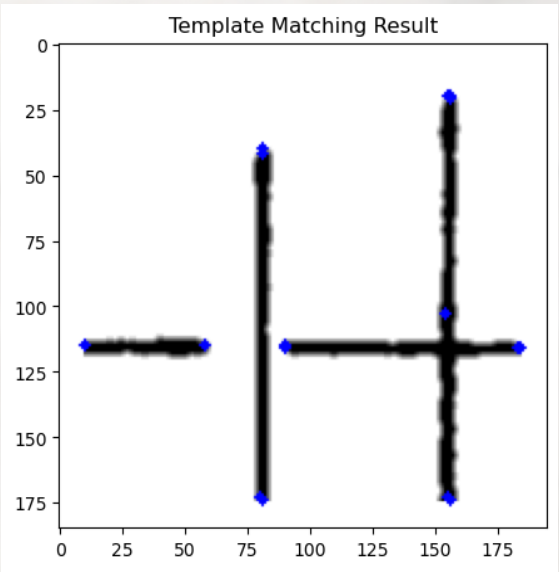


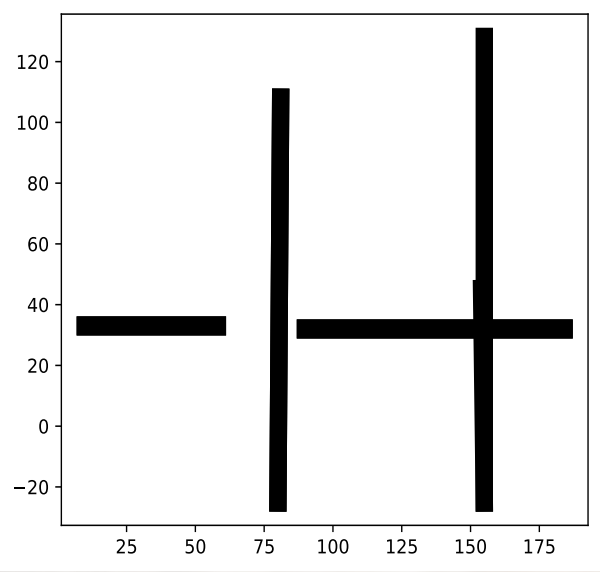


TEMPLATE MATCHING





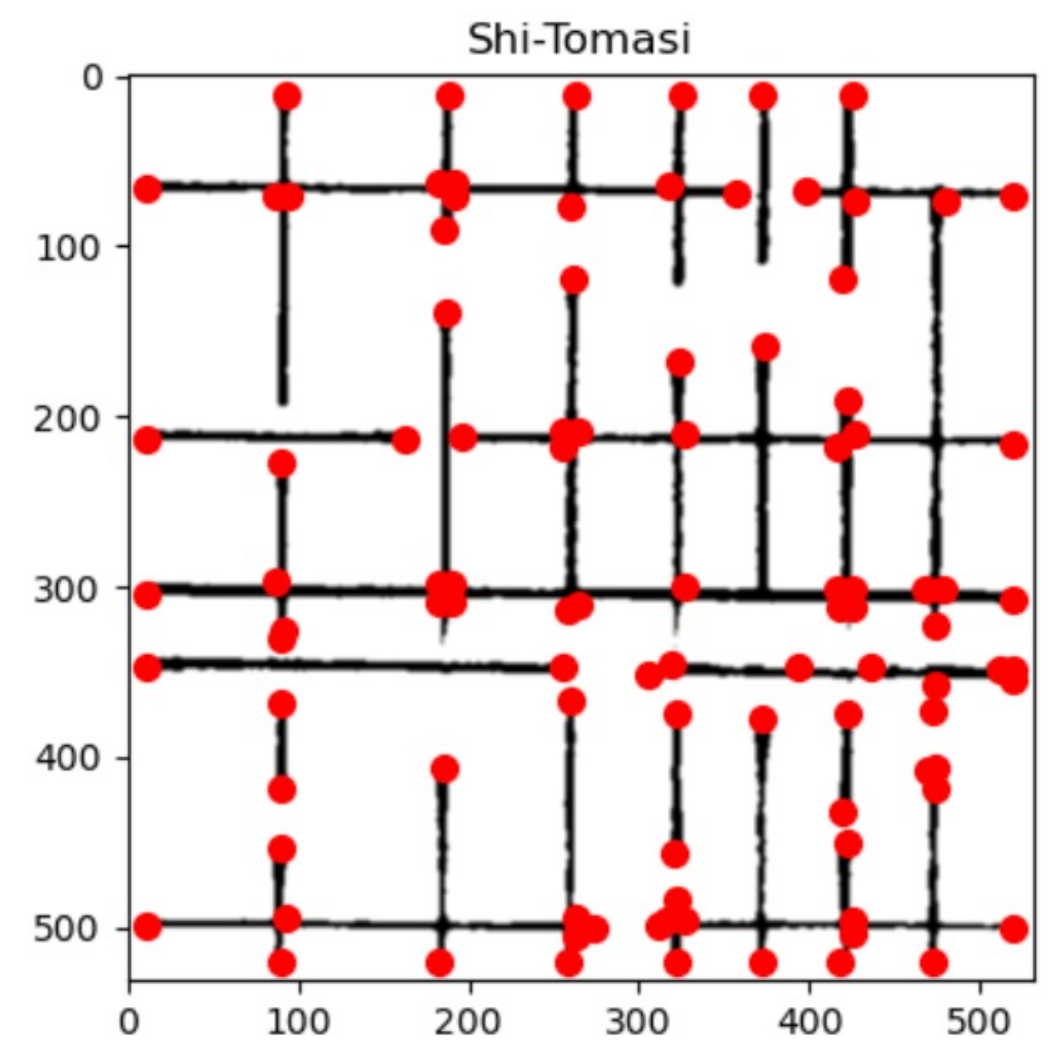


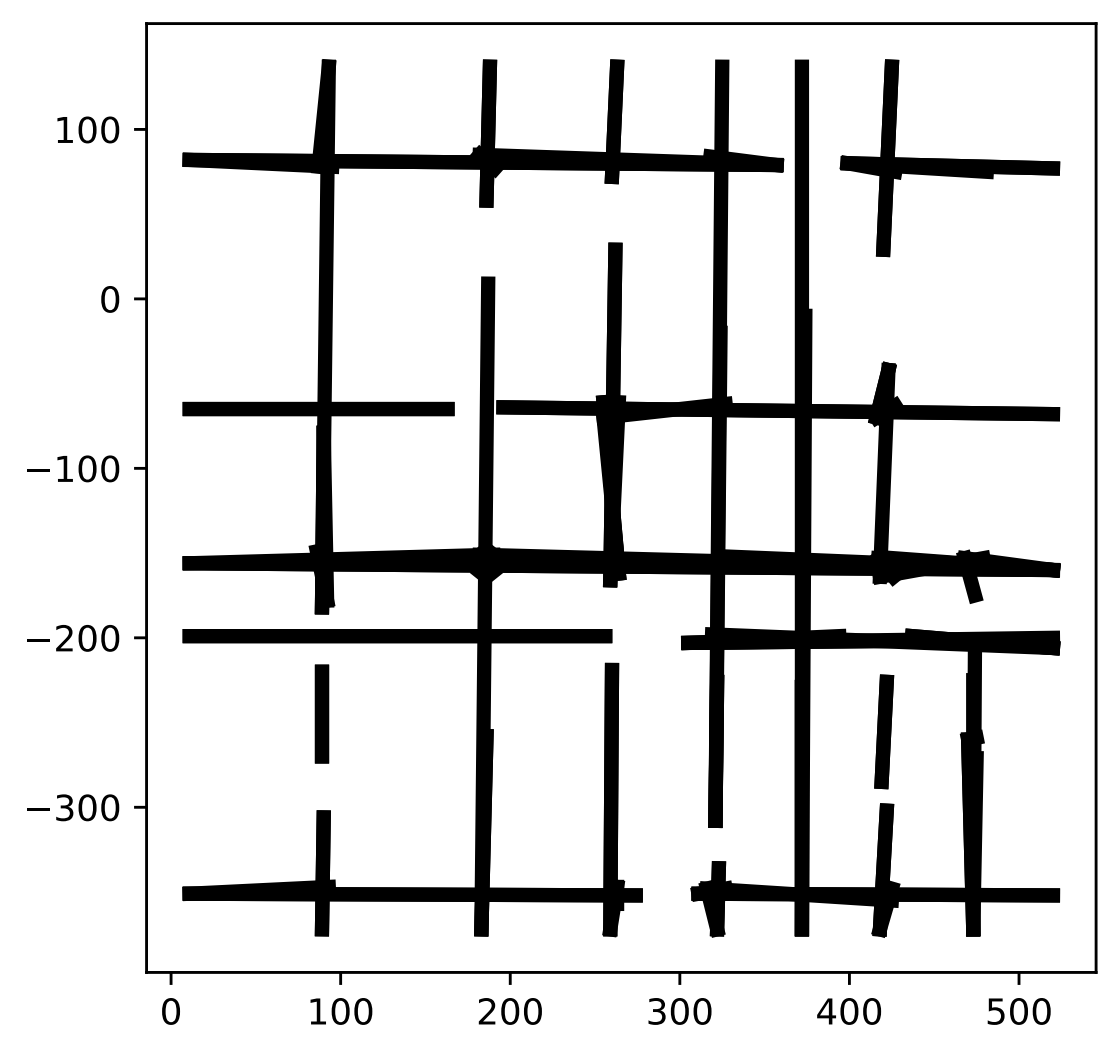


STAGE 2: VECTORIZATION

Comparison

SHI-TOMASI

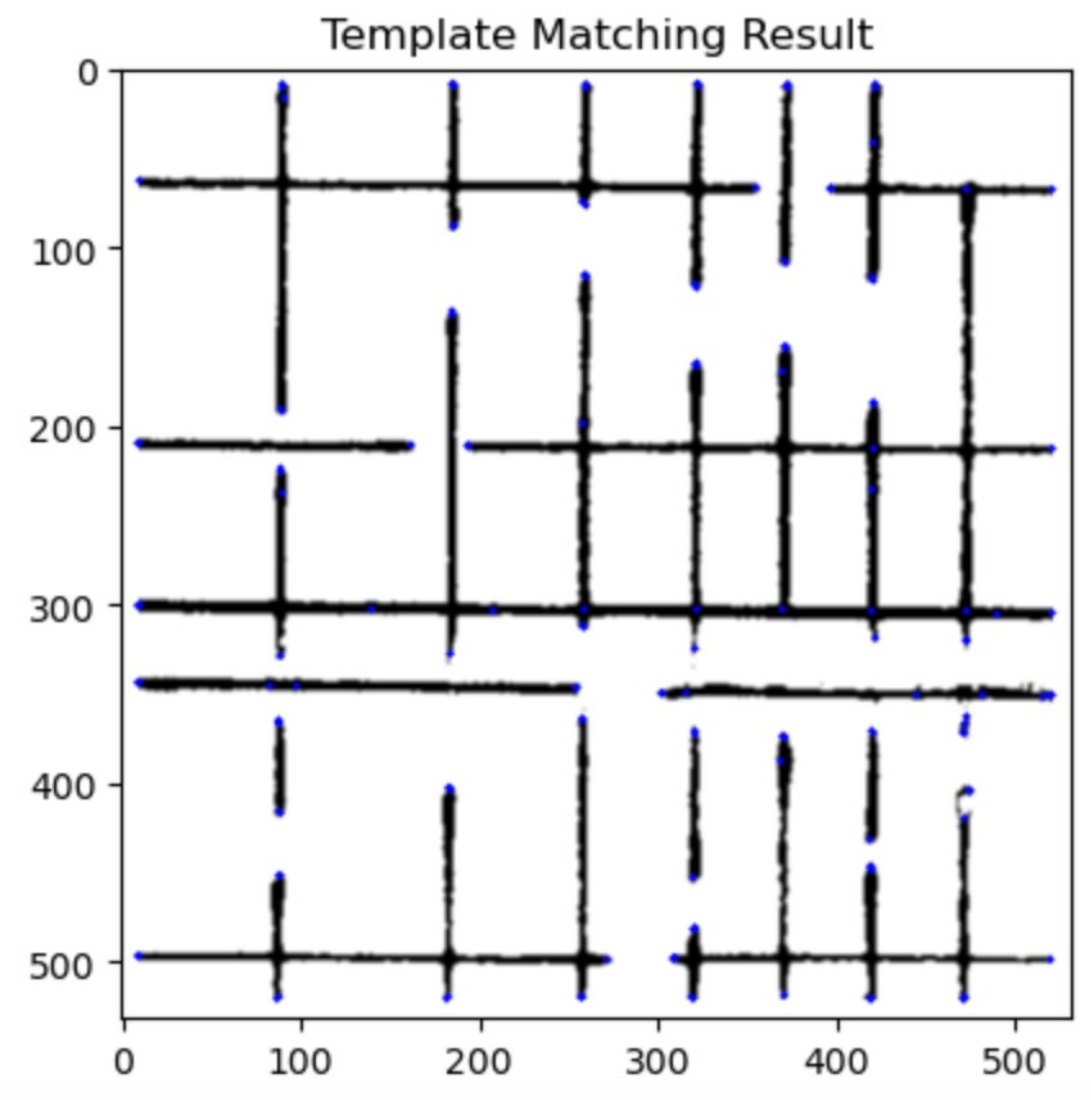


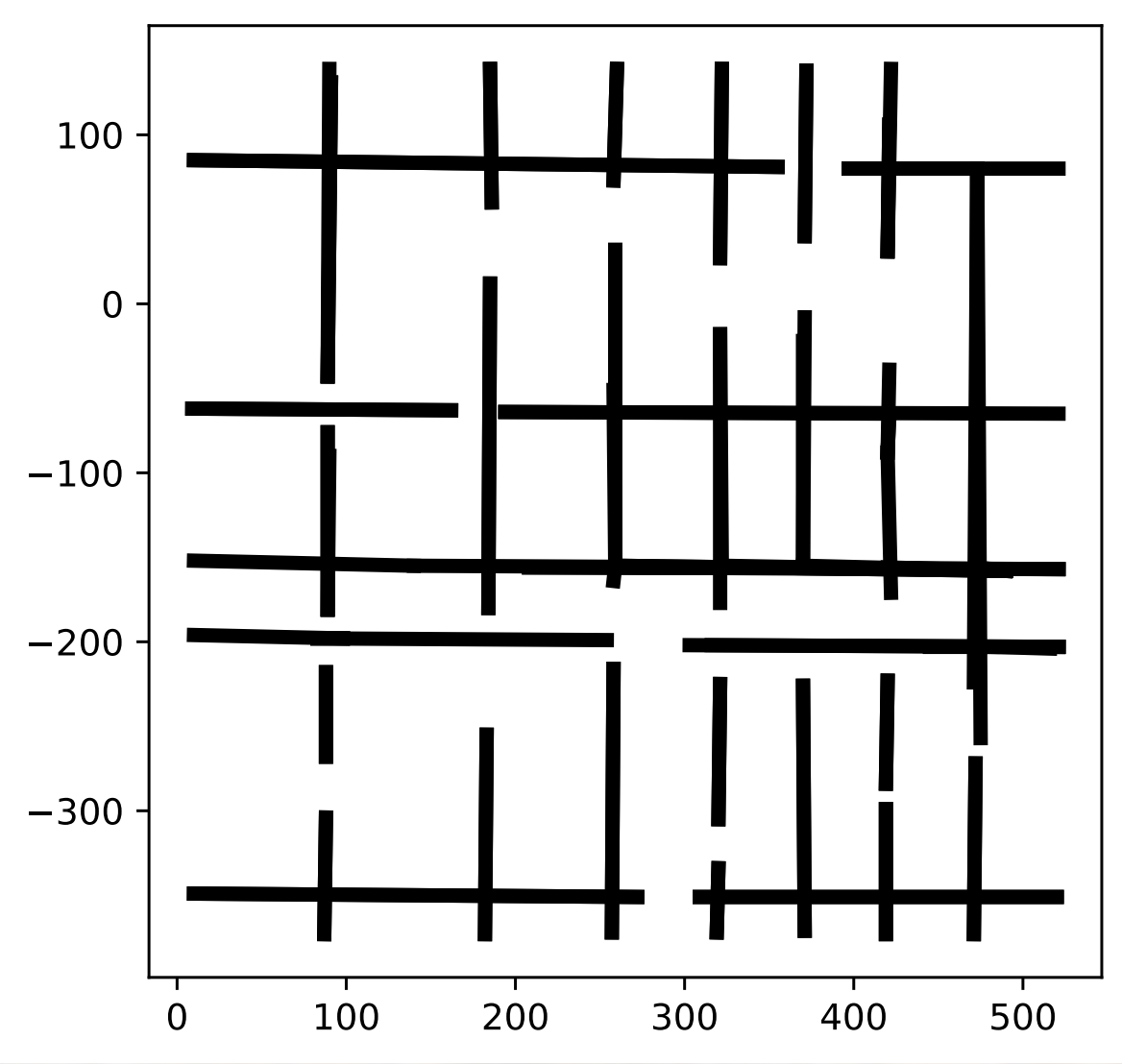


STAGE 2: VECTORIZATION

Comparison

TEMPLATE MATCHING

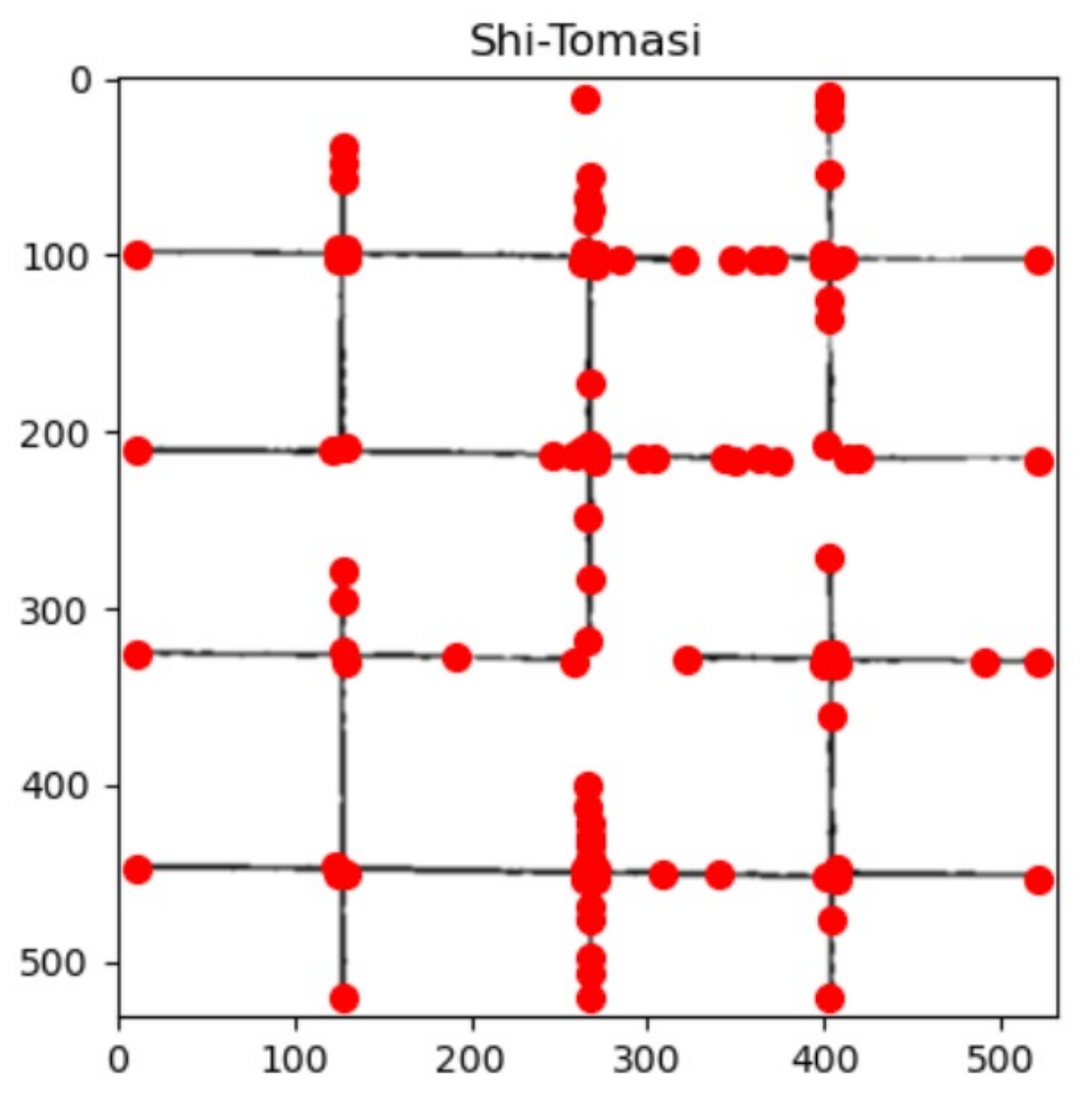


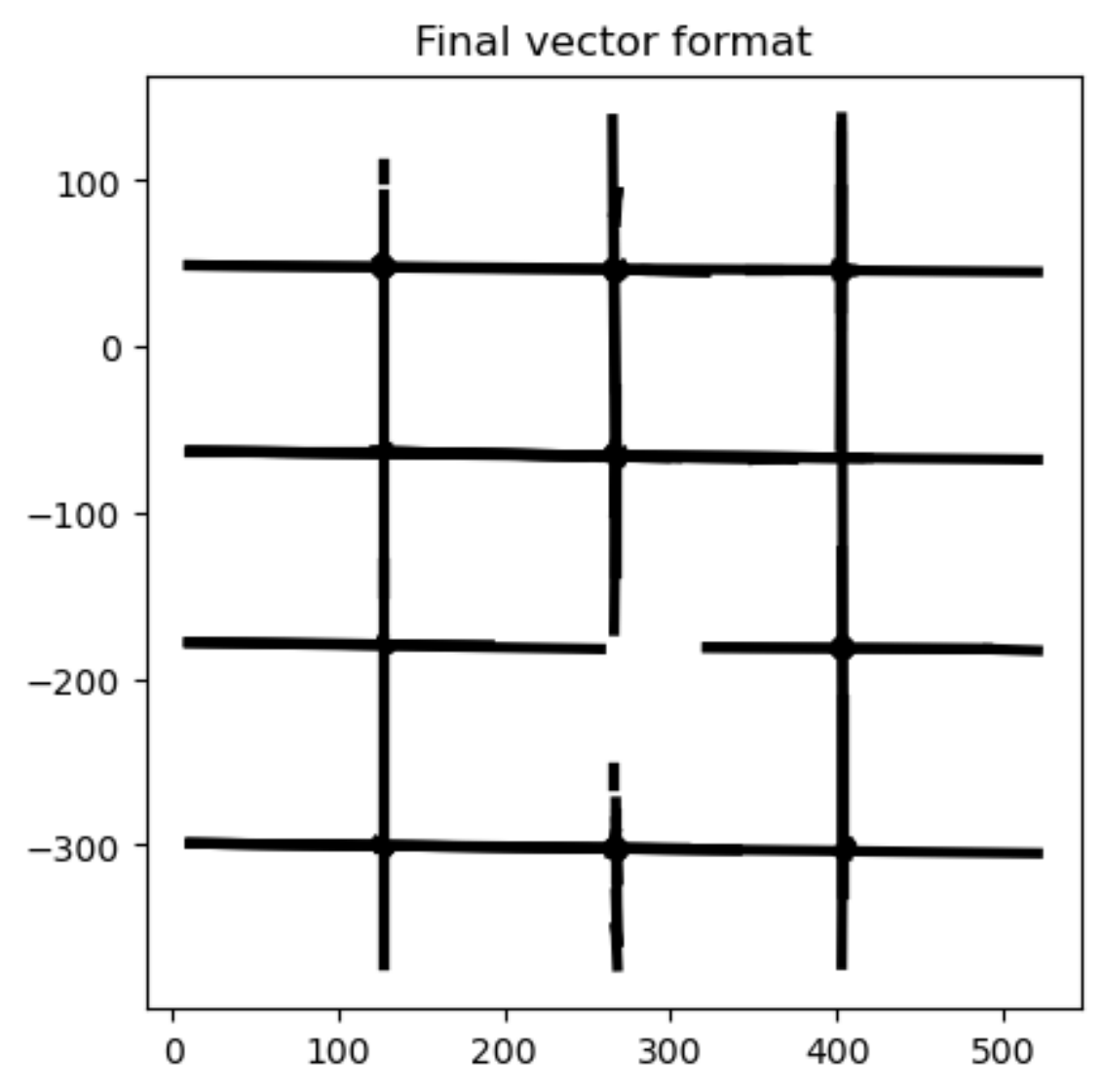


STAGE 2: VECTORIZATION

Comparison

SHI-TOMASI

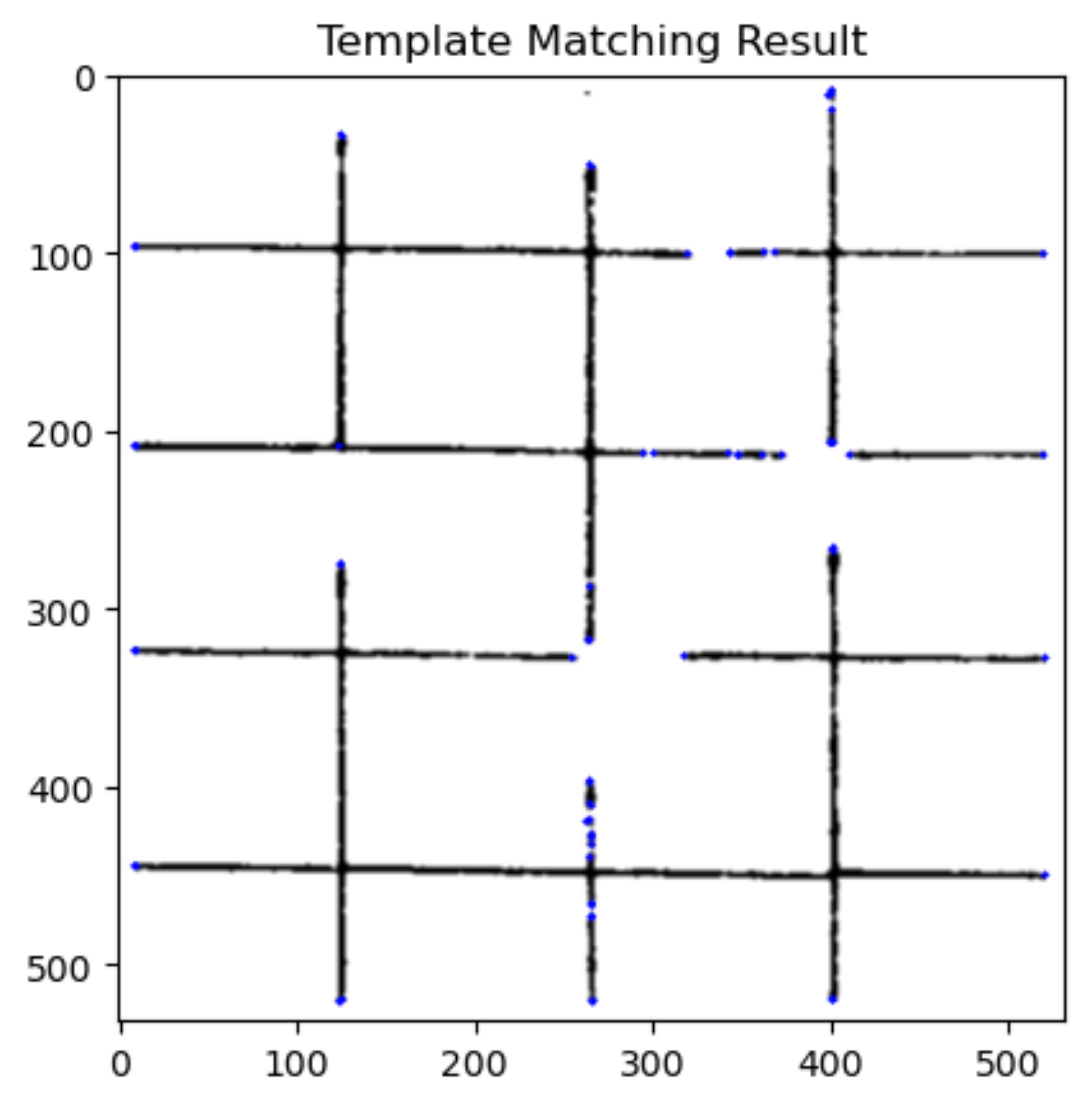


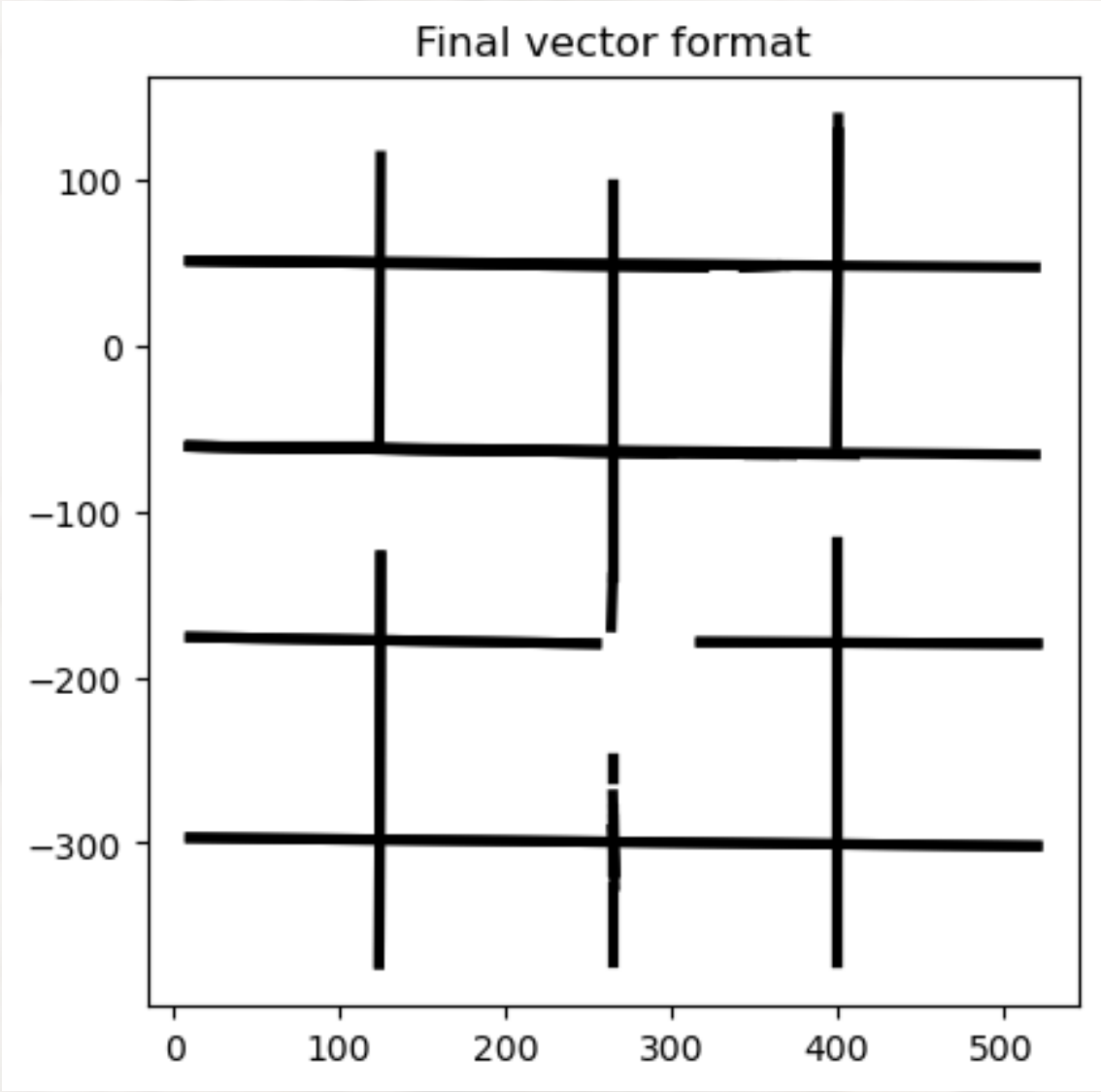


STAGE 2: VECTORIZATION

Comparison

TEMPLATE MATCHING

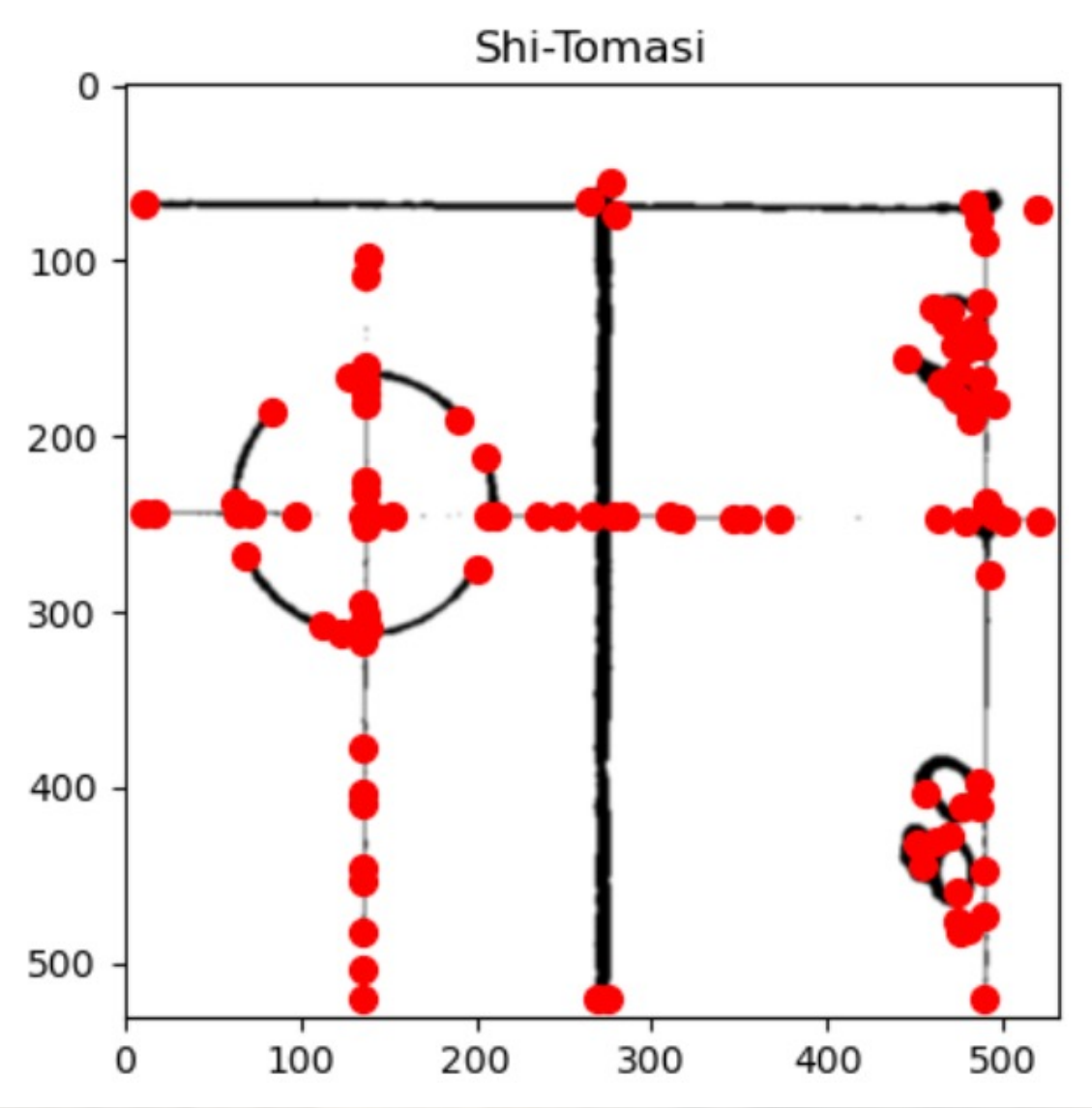


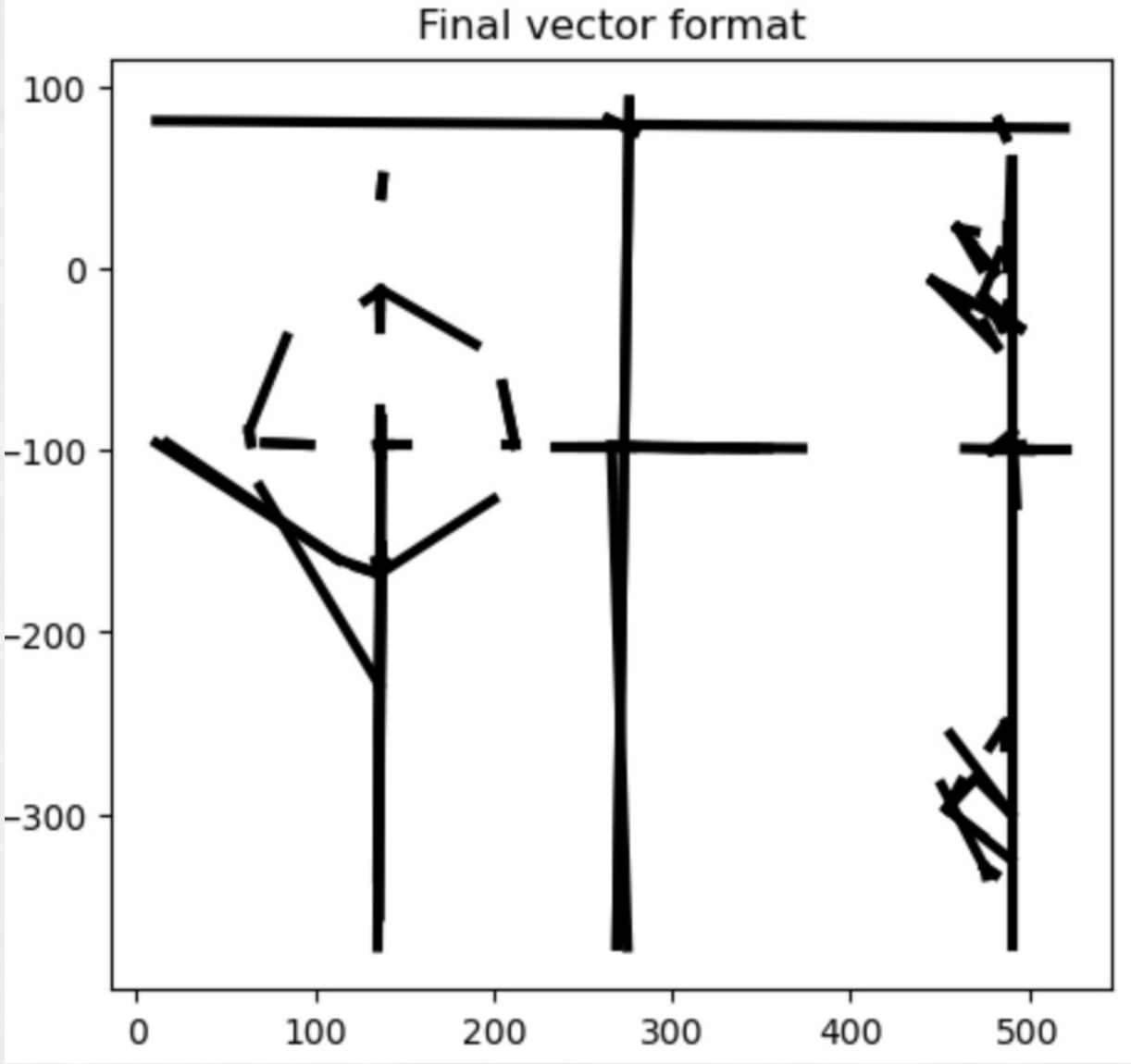


STAGE 2: VECTORIZATION

Comparison

SHI-TOMASI

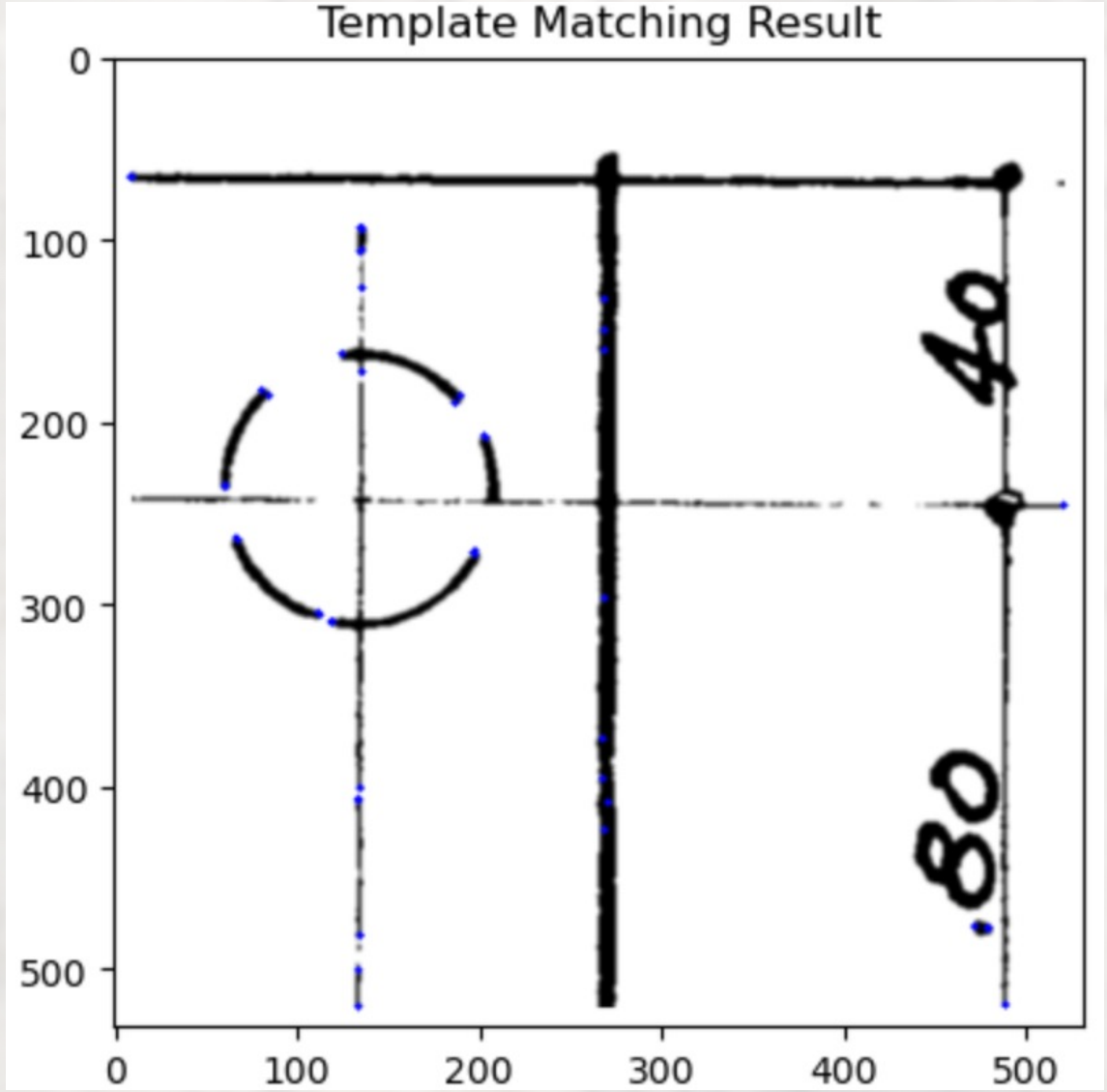


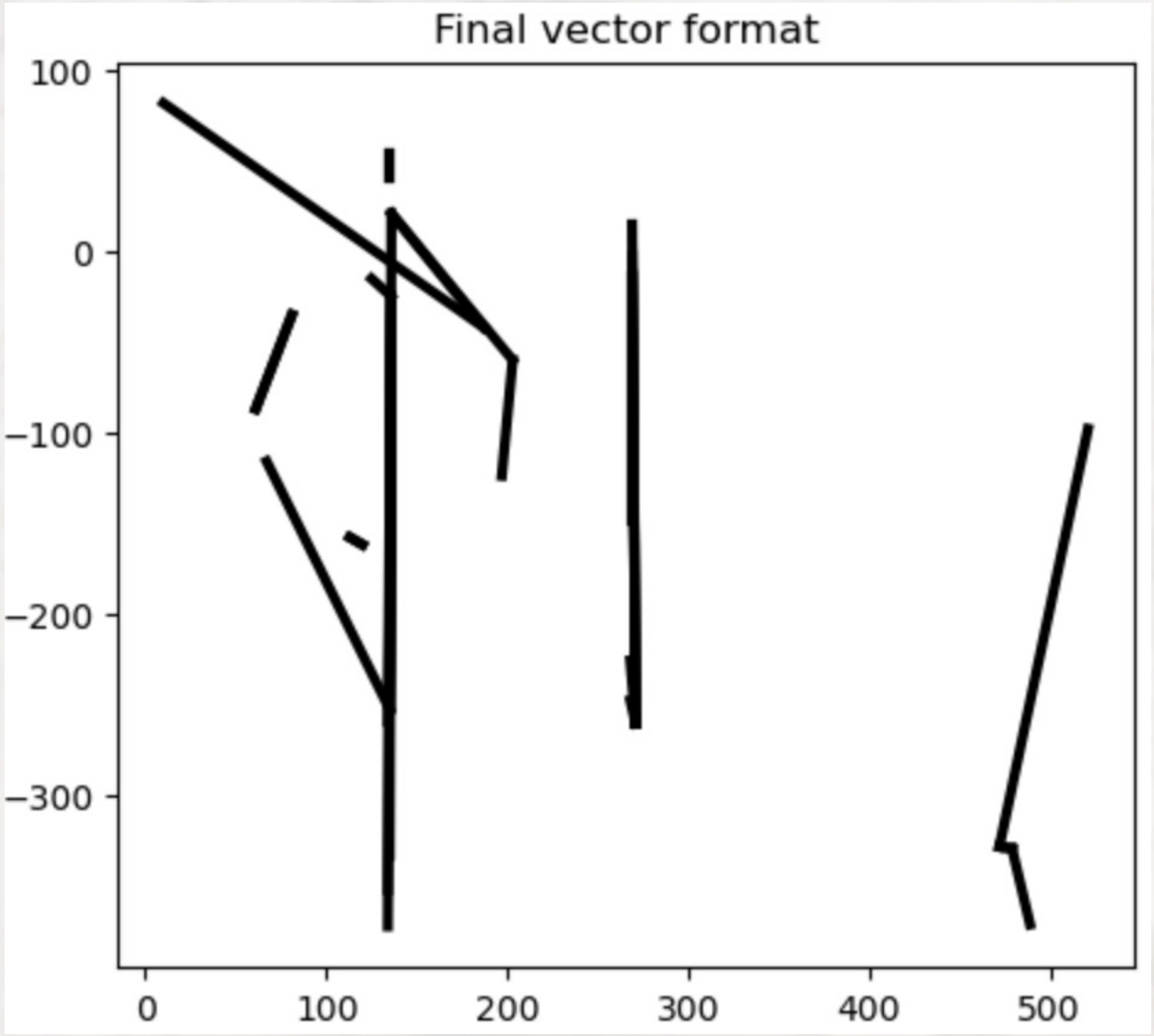


STAGE 2: VECTORIZATION

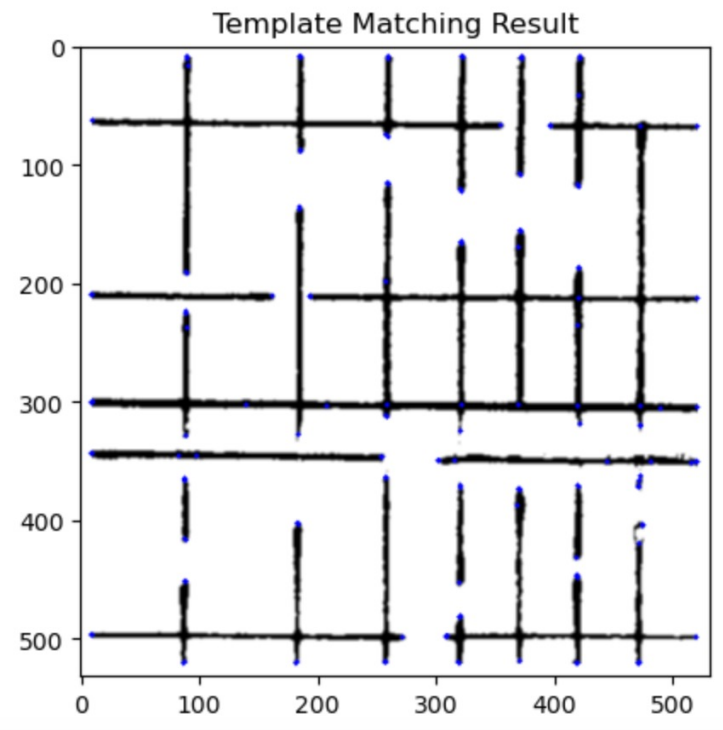
Comparison

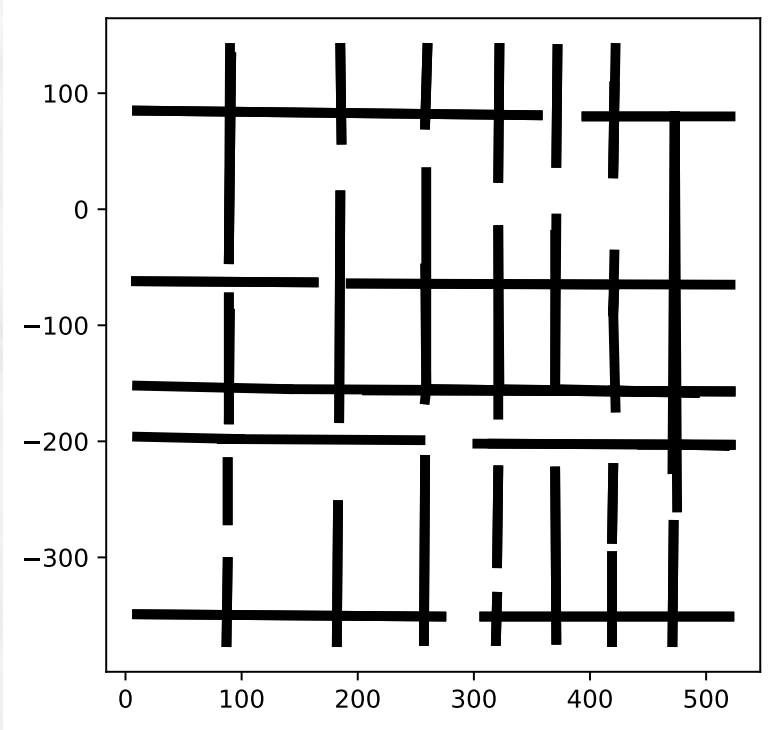
TEMPLATE MATCHING





CONCLUSION





- I was able to vectorize hand-drawn architectural drawings which had never been done before.
- Limited only to straight lines.