MACHINE LEARNING COMPUTER VISION FOR ROBOTIC DISASSEMBLY OF E-WASTE

Bachelor of Engineering in Software Engineering Thesis

Student: Mihail Georgiev

Supervisor: Prof. Maurice Pagnucco

Assessor: Dr. Gwendolyn Foo

BACKGROUND

- Detecting screws in the disassembly scene is a challenging task due to the relatively small size of the screws
- This project explores the Two Stage detection algorithm for screw detection,
- By using a pattern matching screw layout extrapolation
- It builds on an existing base model and improves it by adding additional stage

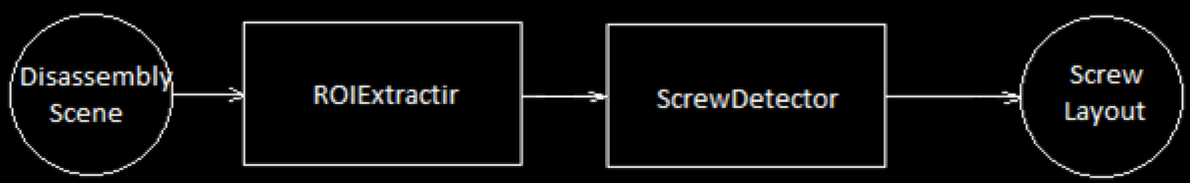
MODEL DESCRIPTION – BASE MODEL

- The base model was proposed in "DCNN-Based Screw Detection for Automated Disassembly Processes" paper, by Erenus Yildiz; Florentin Wörgötter

- It consist of:
  - Region-of-Interest (ROI) extractor, based on circle detection by using the Hough transformation
  - Screw detector, based on two CNN networks (Xception and Inception), pre-trained to detect various types of screws.
  - Implementation based on Open-CV using Python
  - Provided large database of screw images for training the DCNN

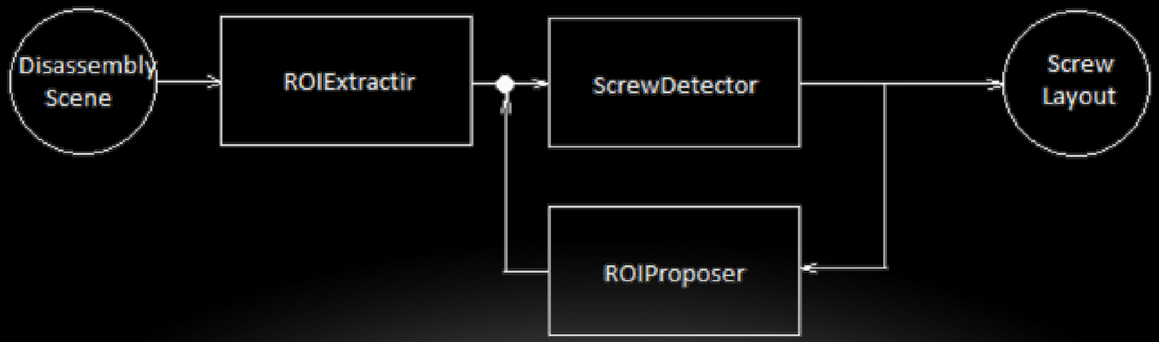
MODEL DESCRIPTION – BASE MODEL

- ROI extractor detects circles in the scene and extracts those parts of the image
- The extracted regions of interest (ROI) are passed to the Detector, consisting of two CNN networks.
- The regions confirmed by the detector are the final screw layout



MODEL DESCRIPTION – TWO-PHASE DETECTION

- The base model copes very well with eliminating the false-positives
- False negatives however prove to be a problem (especially with low quality extractor)
- 2-Phase detection helps coping with the false-negatives by:
  - Extrapolating the screw-layout model and proposing additional regions of interest
  - Passing those additional ROIs to the base model detector for classification
  - Combining the base model and the additional layout into a final layout



PERFORMANCE METRICS

We use the following metrics to evaluate the model performance:

- Precision = True-Positives / (True-Positives + False-Positives)
- Recall = True-Positives / (True-Positives + False-Negatives)
- F1-score = 2 x Precision x Recall / (Precision + Recall)

The main metrics that we use to compare the model performance is the F1-score

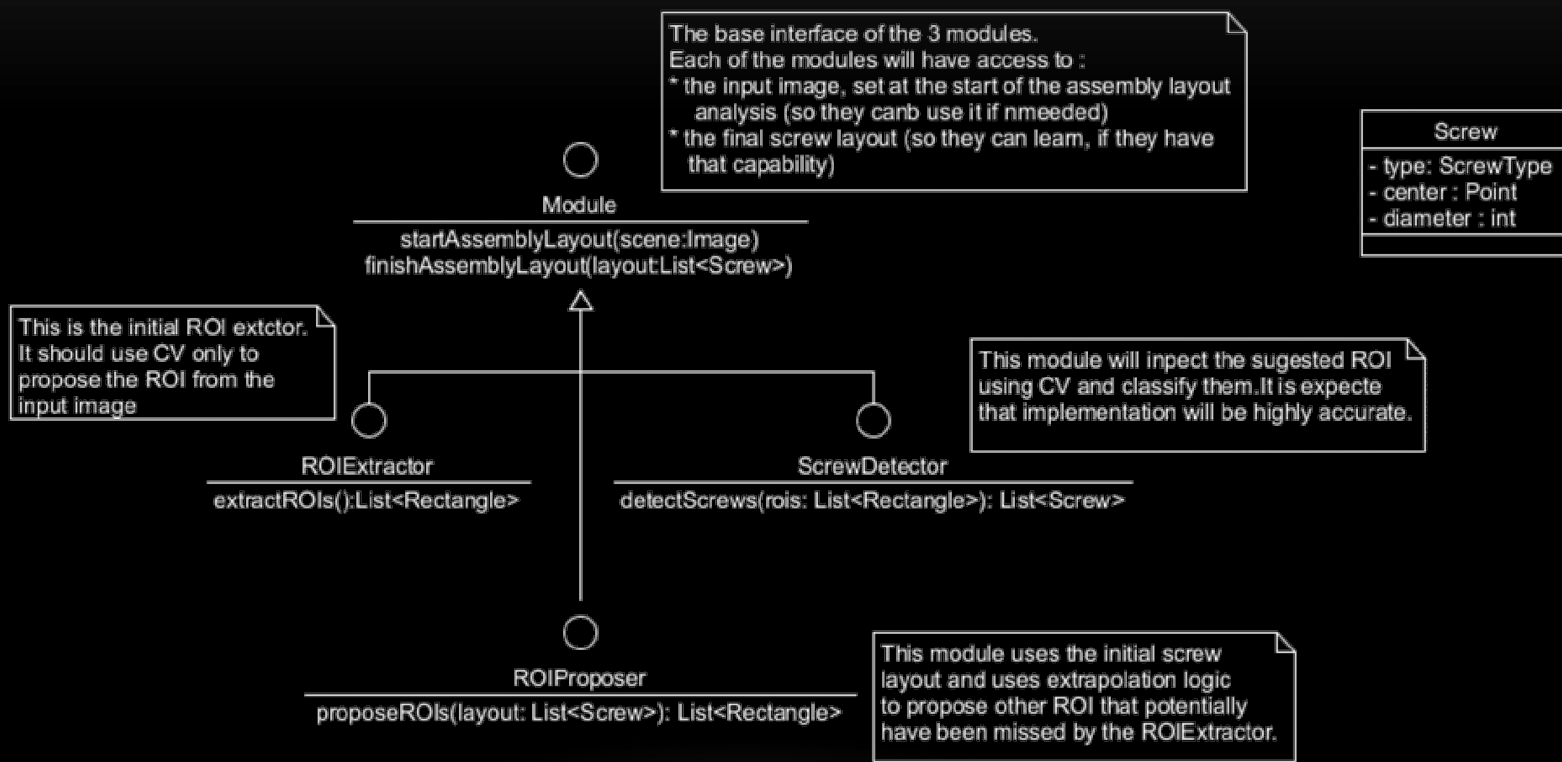
EXPECTED OUTCOMES

- Two-Phase model to improve the base model by:
- Decreasing the number of false negatives
- Increasing the Recall
- Increasing the F1-score

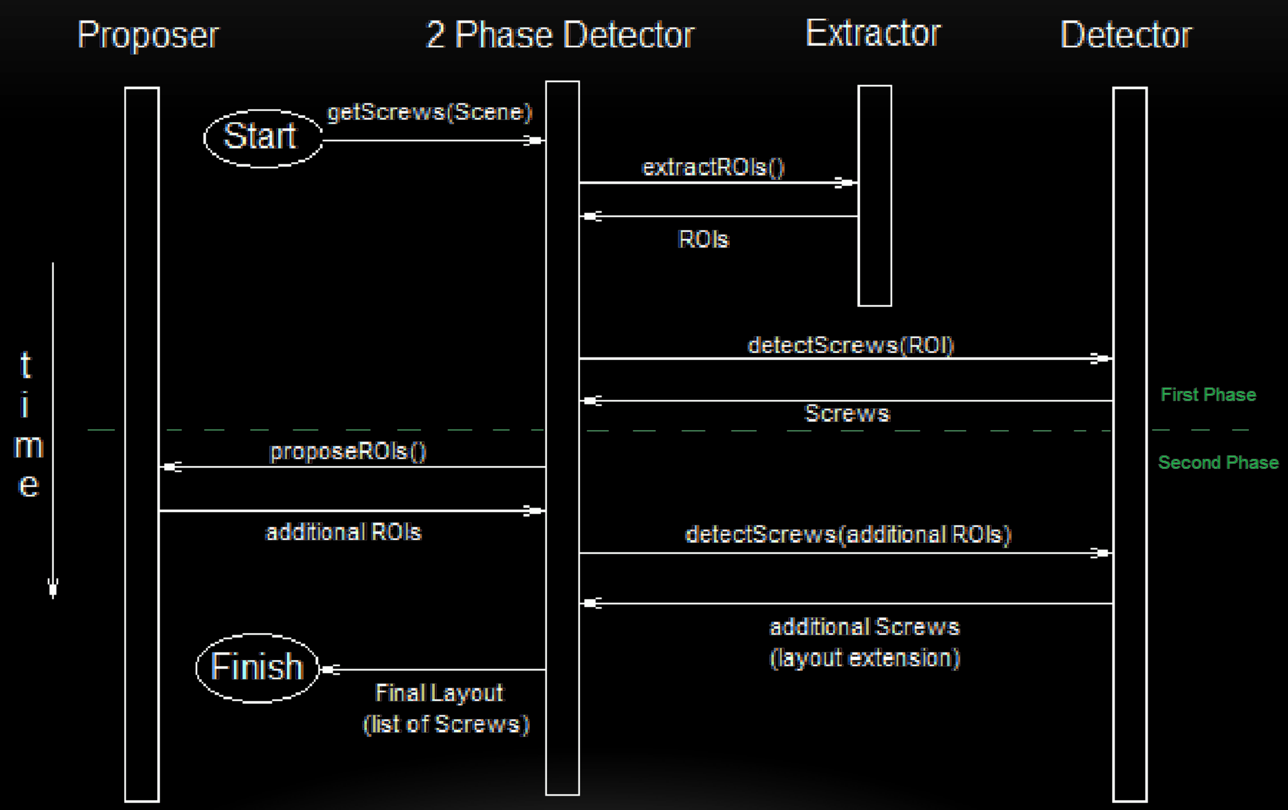
PROJECT SUMMARY

- Defining the components, interfaces and interactions
- Two-Phase detection algorithm
- Pattern Matching Proposer
- Pattern Learning
- Testing Framework
- Test Scene Generation
- Test Results
- Analysis

DEFINING THE COMPONENT INTERFACES

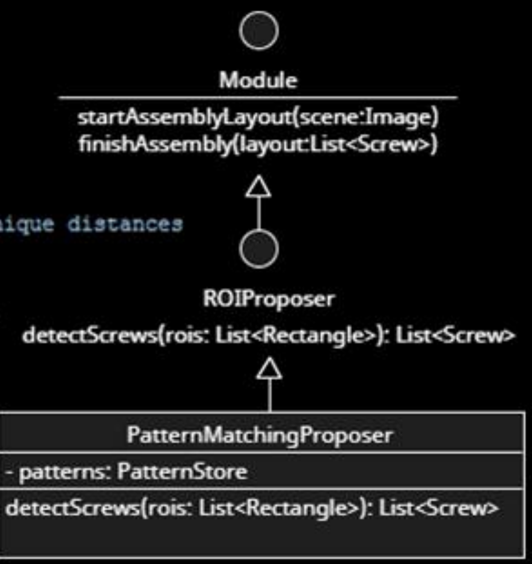


COMPONENT INTERACTIONS
THE TWO-PHASE DETECTION

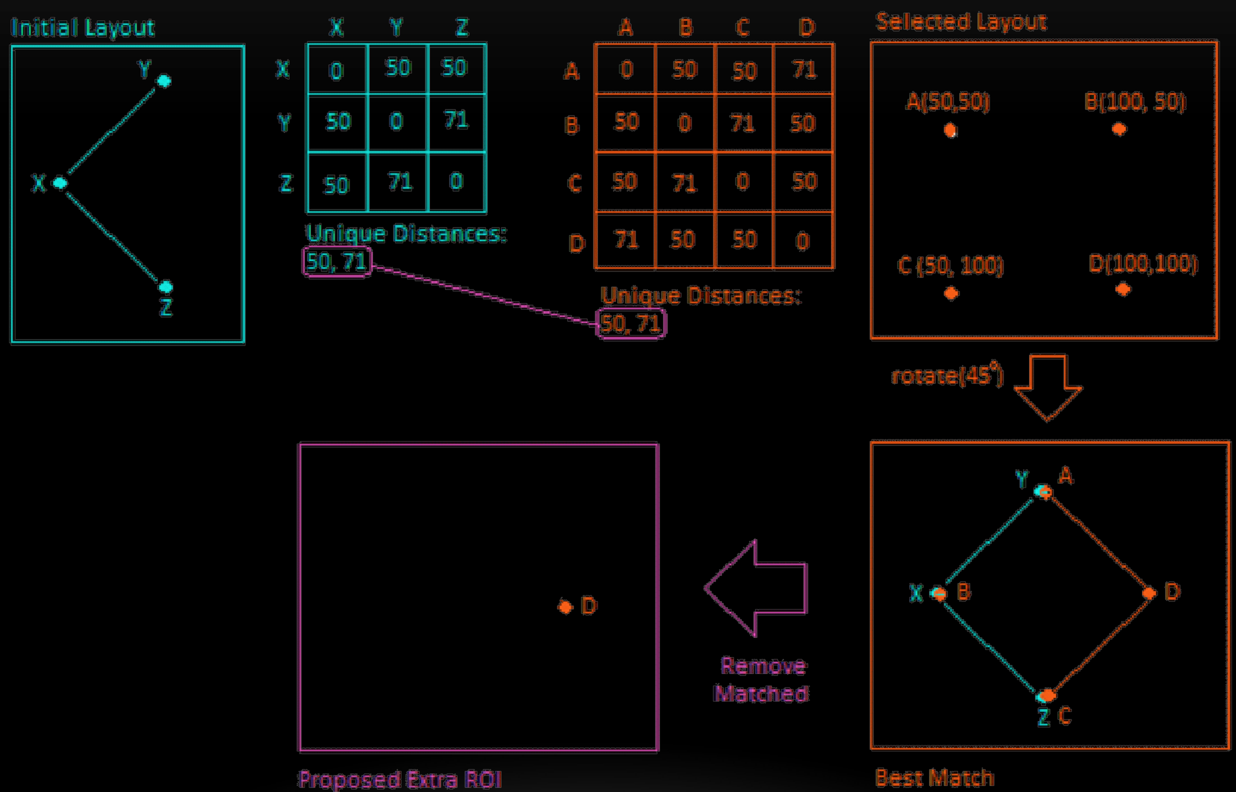


PATTERN MATCHING PROPOSER

```
algorithm PatternMatchingROIProposer
    input:
        initailLayout // list of screw locations detected by the base model
    output:
        proposedROIs // list of proposed extra ROIs
    state
        layoutDatabase // repository of known layouts
begin
    Get the unique distances between the screws in the initailLayour
    selectedLayouts = select from the layoutDatabase the patterns that contain those unique distances
    for each of the selectedLayouts
        bestMatch = get best match with the initail layout
        matchScore = number of matched screws / number of screws in the initalLayout
        if matchScore > best matchScore so far, then
            proposedROIs = bestMatch - initaillayout
        end if
    end for
end
```



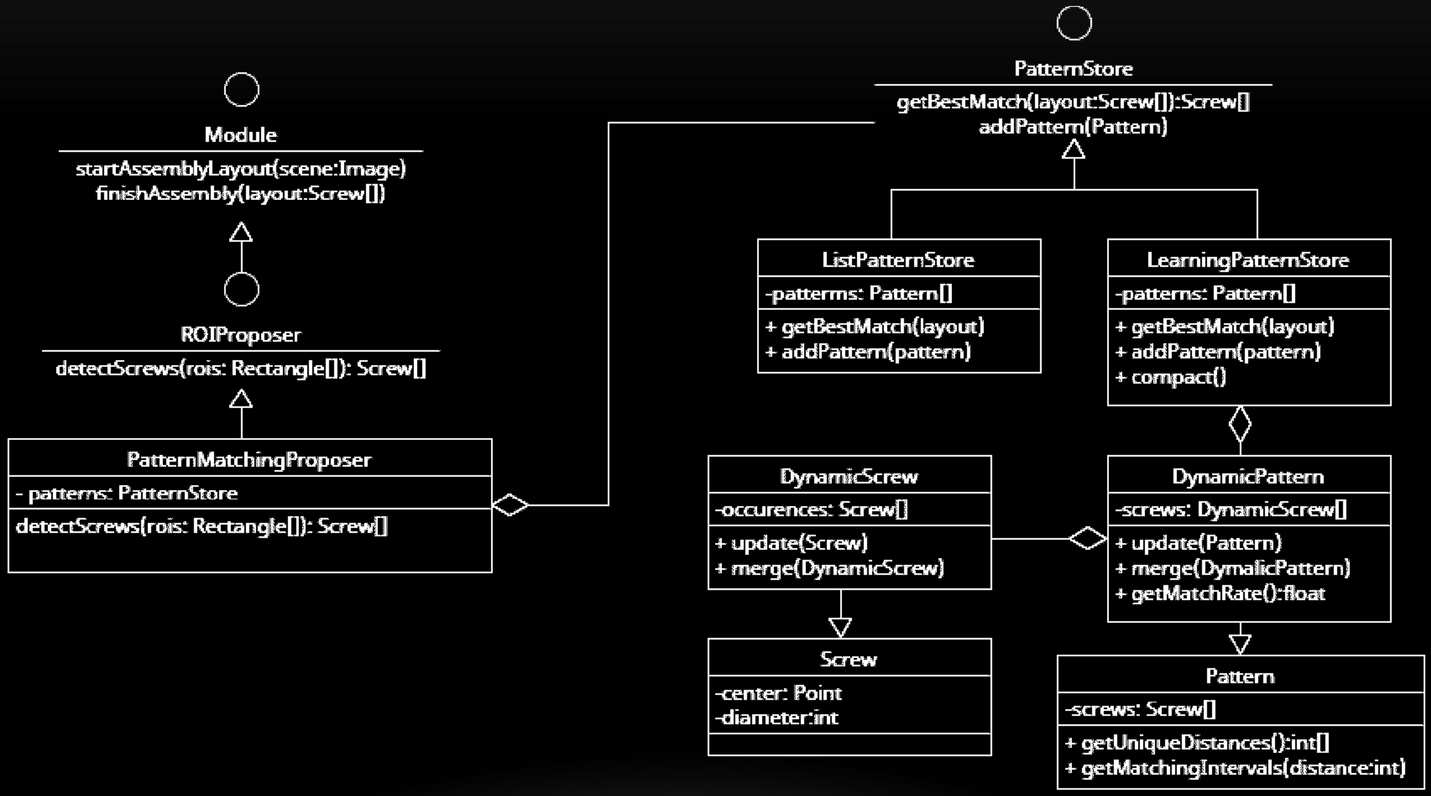
PATTERN MATCHING EXAMPLE



	X	Y	Z
X	0	50	50
Y	50	0	71
Z	50	71	0

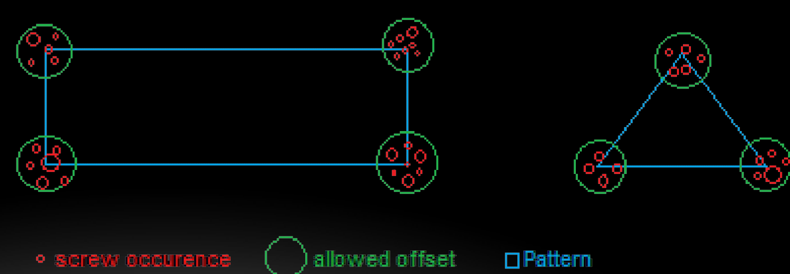
	A	B	C	D
A	0	50	50	71
B	50	0	71	50
C	50	71	0	50
D	71	50	50	0

LEARNING


Module
startAssemblyLayout(scene:Image)
finishAssembly(layout:Screw[])
ROIProposer
detectScrews(rois: Rectangle[]): Screw[]
PatternMatchingProposer
- patterns: PatternStore
detectScrews(rois: Rectangle[]): Screw[]
PatternStore
getBestMatch(layout:Screw[]):Screw[]
addPattern(Pattern)
ListPatternStore
-patterms: Pattern[]
+ getBestMatch(layout)
+ addPattern(pattern)
LearningPatternStore
-patterns: Pattern[]
+ getBestMatch(layout)
+ addPattern(pattern)
+ compact()
DynamicScrew
-occurences: Screw[]
+ update(Screw)
+ merge(DynamicScrew)
DynamicPattern
-screws: DynamicScrew[]
+ update(Pattern)
+ merge(DymalicPattern)
+ getMatchRate():float
Screw
-center: Point
-diameter:int
Pattern
-screws: Screw[]
+ getUniqueDistances():int[]
+ getMatchingIntervals(distance:int)

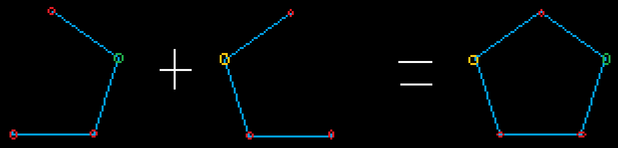
LEARNING ALGORITHM

```
algorithm MemoriseLayout:
    input:
        patternStore // List of memorised patterns
        newLayout    // Layout (list of screws) to be memorised
        mergeThreshold  // minimal match score neede for two patterns to be considered same
    output:
        patternStore // the updated list of patterns
begin
    bestMatchPattern = Find best matching pattern from the patternStore
    bestMatchScore = number of matched screws / total number of screws in the newLayout

    if bestMatchScore > mergeThreshold,
    then update the bestMatchScore with newLayout,
    else create a new pattern from the newLayout and add it to patternStore.

    Periodically, compact the patternStore.
end
```

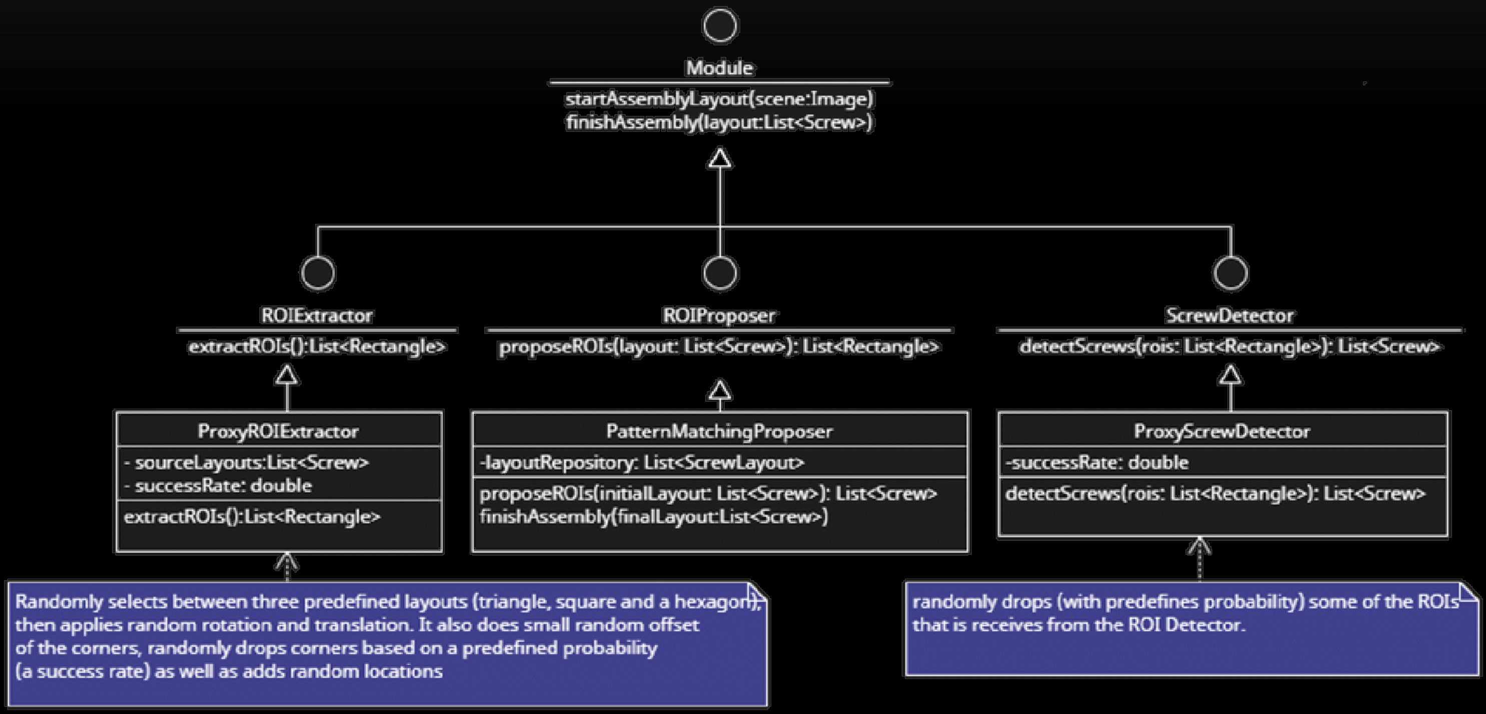


COMPACTING THE PATTERN STORE

```
algorithm CompactPatternSctore:
    input:
        patternStore    // is list of patterns
        forgetThreshold // is minimal usage score for pattern to be retained
        mergeThreshold  // is the minimum match score for patterns to be merged
    output:
        patternStore // the compacted patternStore (containing same or less patterns)
begin
    for each pattern in patternStore:
        if usage ratio < forgetThreshold, then remove it from the patternStore.

        bestMatch = Find the best match among the other patterns in the patternStore.
        if the bestMatchScore > mergeThreshold, then merge the two patterns.
end
```



TESTING FRAMEWORK



TESTING – COMPONENT QUALITY SENSITIVITY

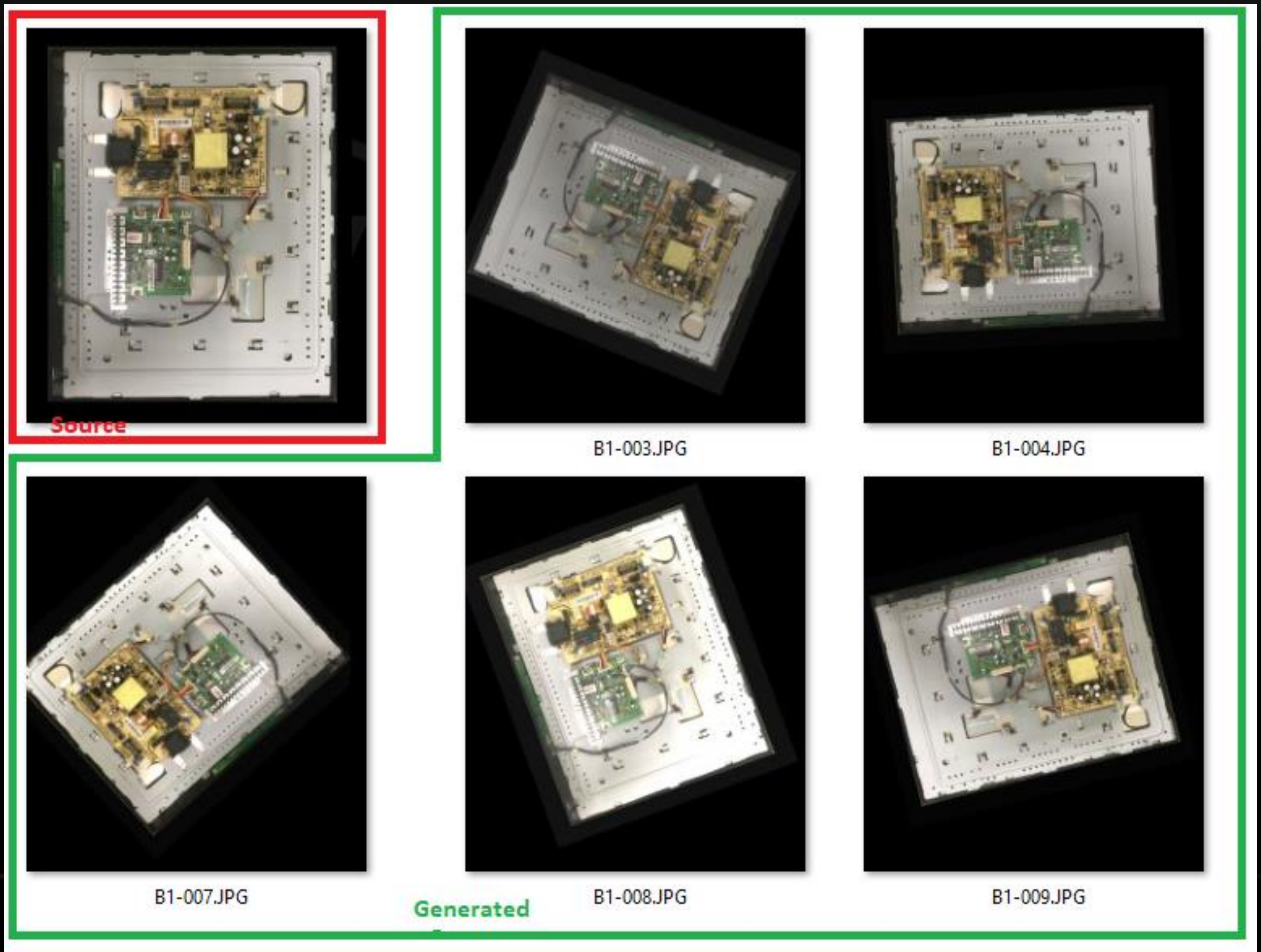
- Tested the performance of the Pattern Matching Proposer with testing framework
- Over 100 scenes, generated over 3 base patterns: rectangle, triangle and hexagon
- Using gradually improving quality of extractor and detector
- Used F1-score as main performance measure
- Two-Pass model consistently outperformed the Base Model
- Lower quality of the base model resulted in lower gain

ROI Extractor Success rate	Screw Detector Success rate	Base Model F1-score	Two Pass F1-score	Gain
0.5	0.6	0.34	0.35	3%
0.6	0.8	0.56	0.64	14%
0.7	0.9	0.71	0.79	11%

GENERATING TEST SCENES

- Selected several distinct disassembly scene photos
- Removed the perspective
- Removed the background
- Manually annotated them using https://www.makesense.ai/
- Created a scene generator to apply random:
  - translation
  - rotation
  - slight contrast and lighting changes
- Created a Scene Source to combine the base scenes and the scene generator to provide infinite sequences

SAMPLE SCENES

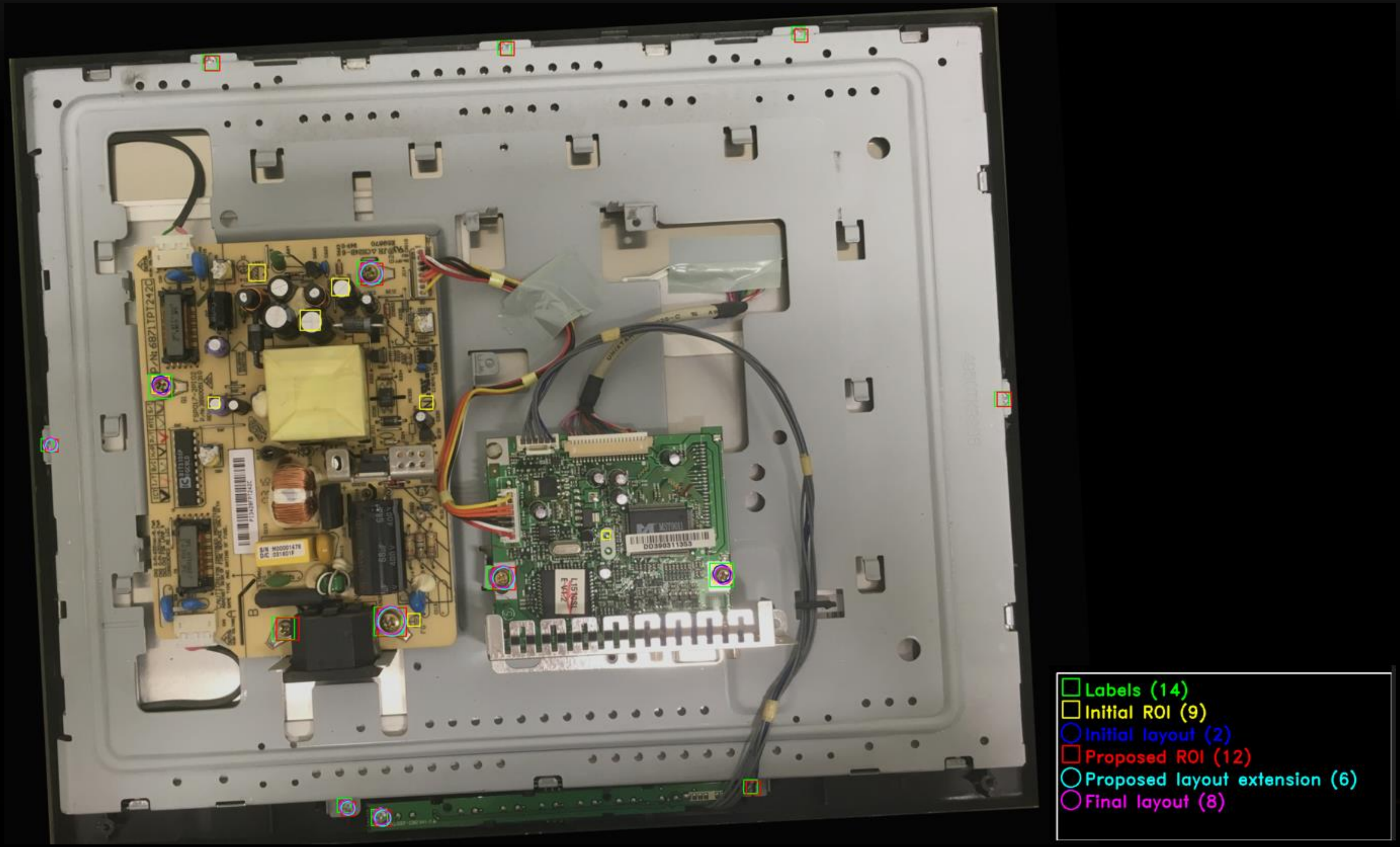


TESTING – PRE-LOADED PATTERNS

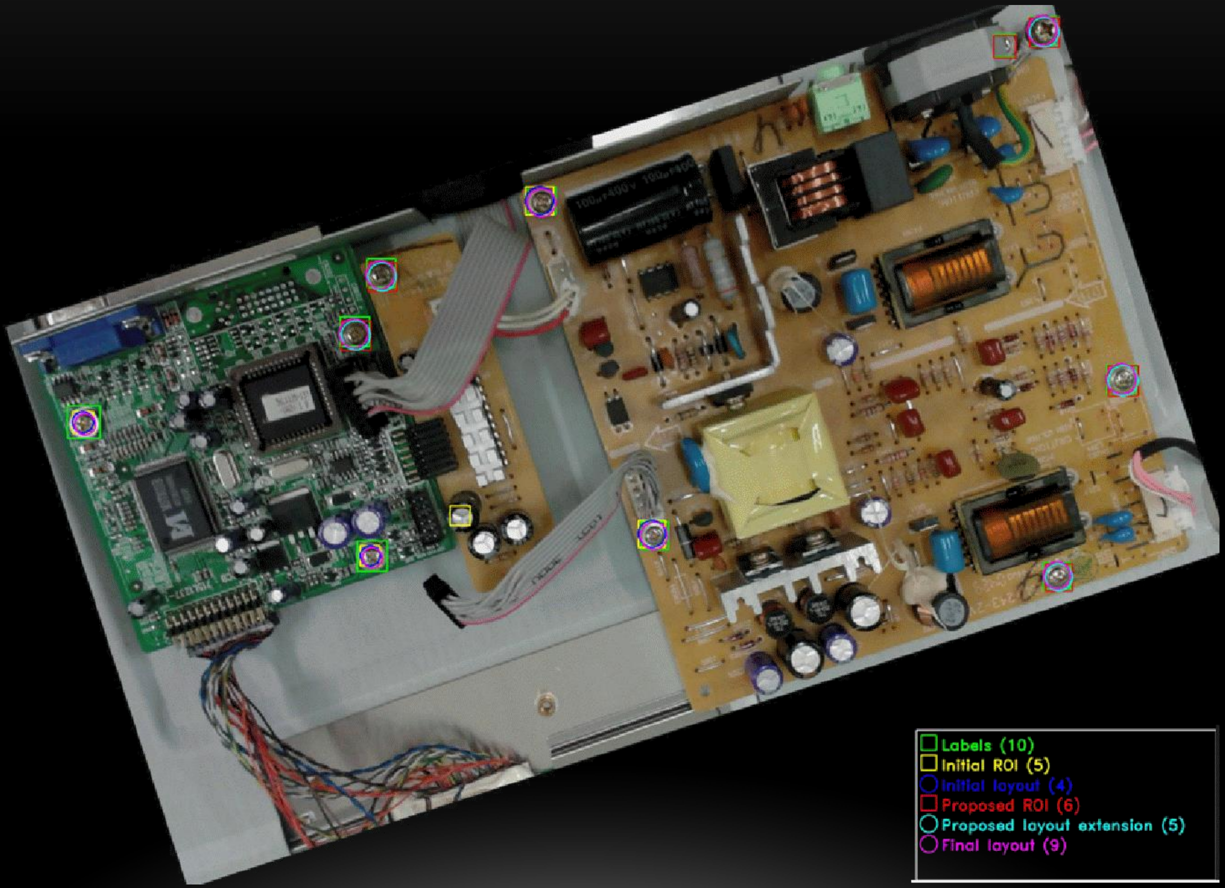
- Motivation – to test the 2-phase model best case scenario
- All patterns are already in the memory
- Used 100 generated scenes over 3 distinct base layouts
- 2-Phase model achieved 63% improvement on the primary metric (F1-score)

	Precision	Recall	F1-score	Offset
Base Model	0.98	0.28	0.43	3.08
2-Phase Model	0.93	0.61	0.70	4.07
Performance Gain	-5%	117%	**63%**	-32%

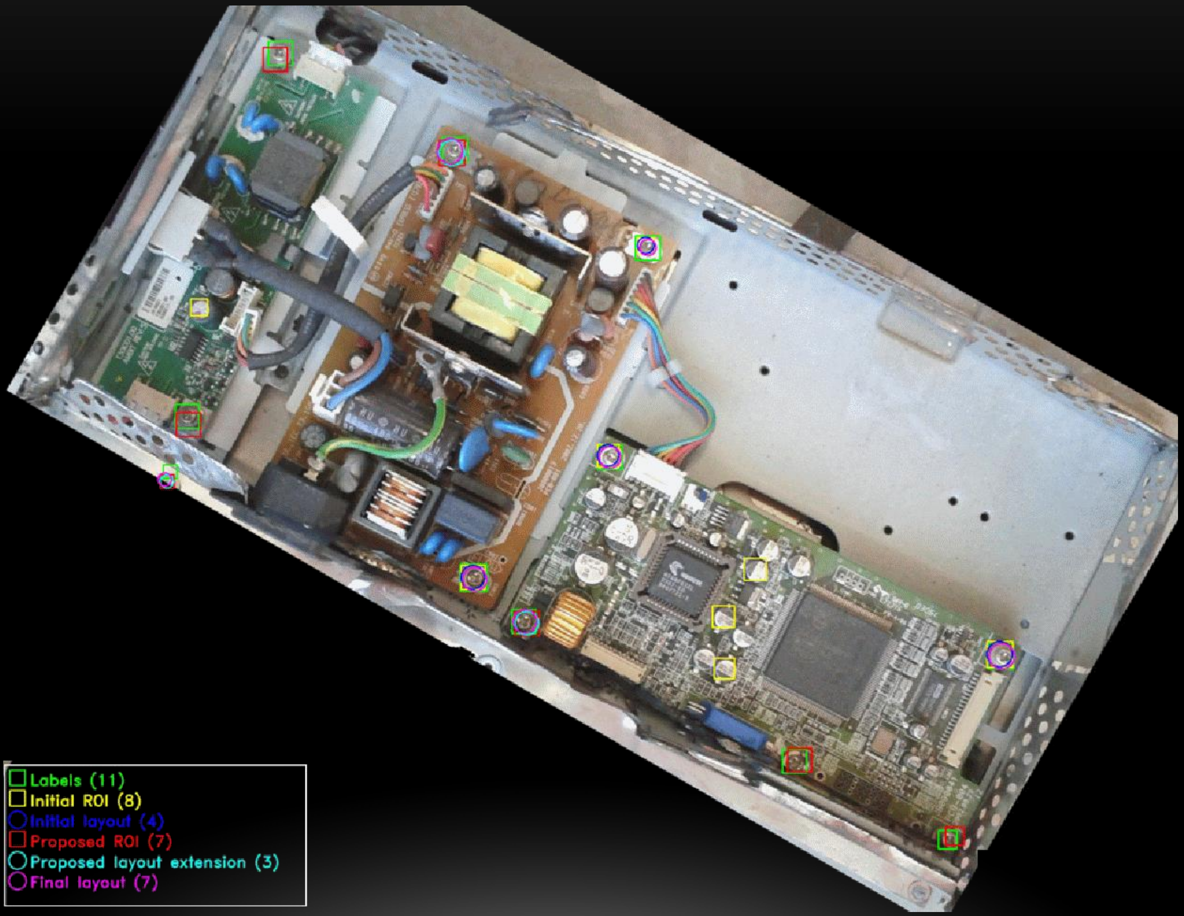
SAMPLE TEST SCENES



SAMPLE TEST SCENES



SAMPLE TEST SCENES



TESTING – LEARNING STORE

- Motivation – to test the effectiveness of the 2-phase model learning store
- Started with empty pattern store
- Used 100 generated scenes over 3 distinct base layouts
- 2-Phase model achieved 16% improvement on the primary metric (F1-score)

	Precision	Recall	F1-score	Offset
Base Model	0.99	0.28	0.43	3.24
2-Phase Model	0.97	0.34	0.50	3.40
Performance Gain	-2%	21%	**16%**	-5%

ANALYSIS - QUESTIONS

- Why do we have negative accuracy and precision gain ?
- Why the learning proposer gain was inferior to the one with preloaded patterns ?
- Can the 2-phase model have a negative gain (perform worse than base model) ?
- If negative gain is possible, how can we prevent it ?

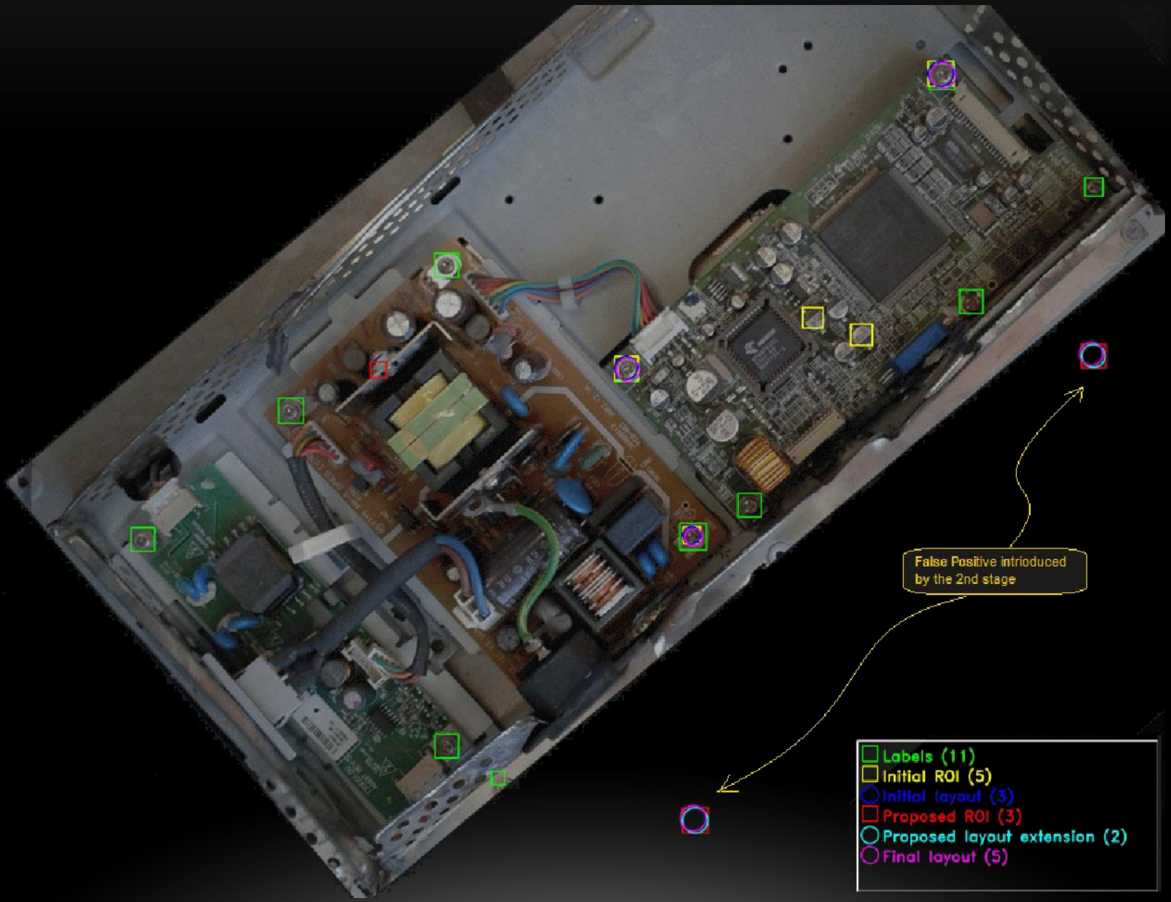
ANALYSIS – PRECISION DECREASE

- The precision decrease can be explained by the 2^{nd}-phase adding false-positives:

Precision = True Positives / True Positives + False Positives

- How can that happen?
- Incorrect pattern match, introduces wrong ROIs.
- Then, some of those ROIs are incorrectly detected as screws (false positives) by the detector (because the detector is not perfect)
- Indeed we do have such cases …

ANALYSIS – PRECISION DECREASE

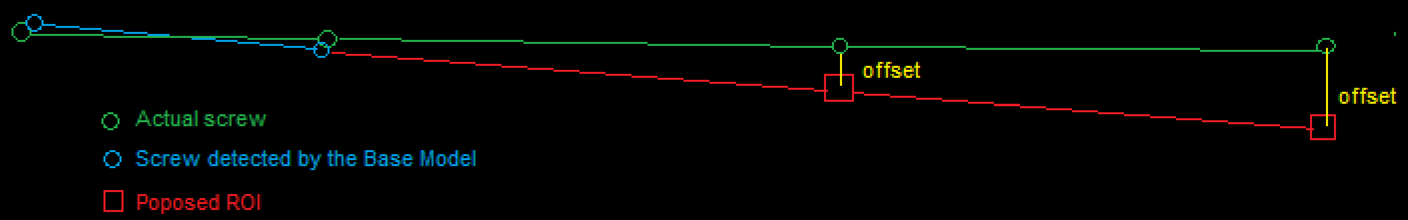


ANALYSIS – ACCURACY DECREASE

- Accuracy decrease can be explained by 2nd phase adding screws that are on average further away from the actual location than the base model ones

  Accuracy = average offset of the detected screw centres from the actual screw centre

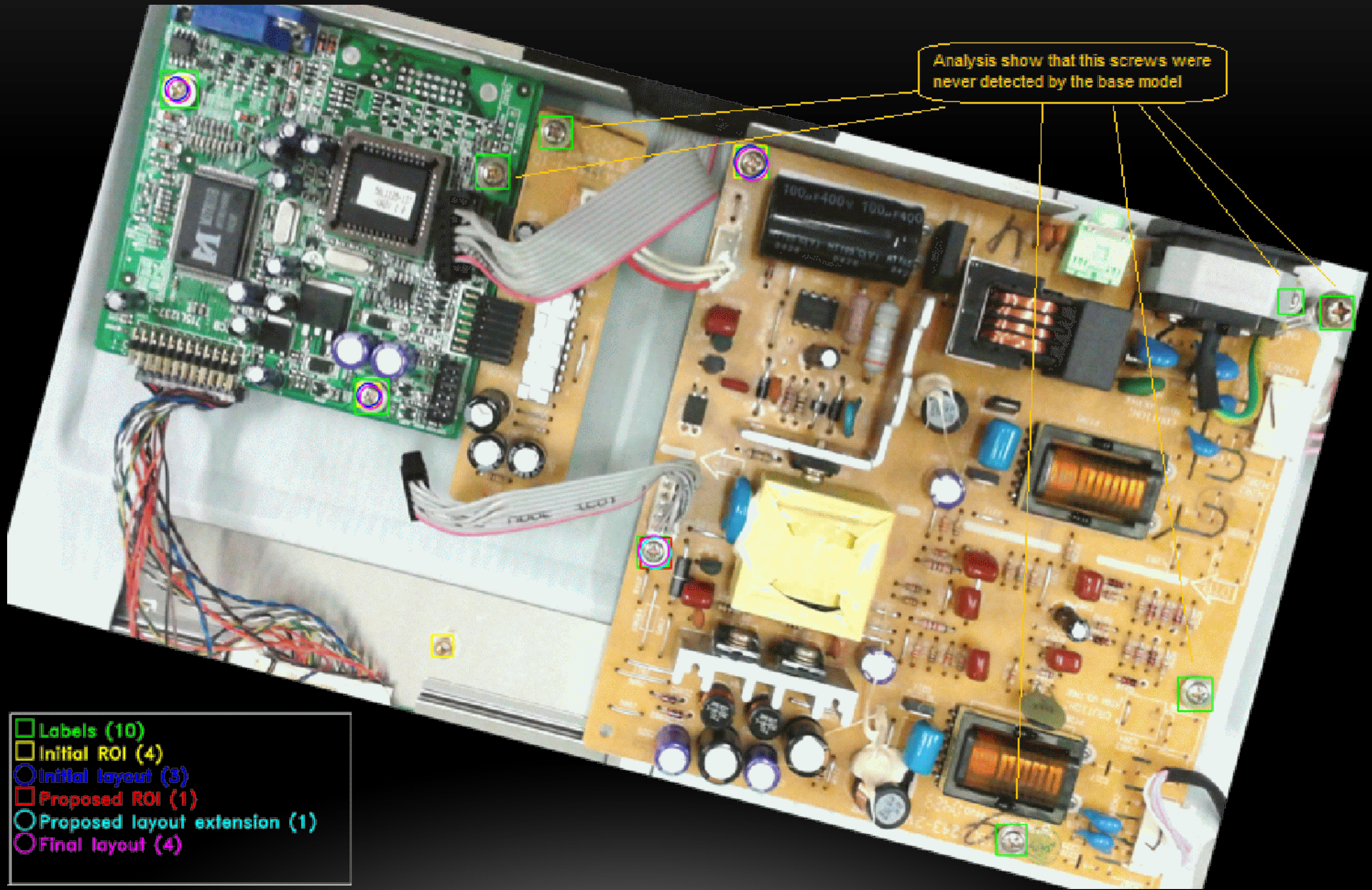
- The base model layout that is used to match a pattern have some accuracy error
- That error gets amplified in the offsets of the proposed ROIs
- The further the proposed ROI is from the base layout, the larger the error is.



ANALYSIS – LEARNING PROPOSER GAIN

- Why is the learning proposer gain inferior than the proposer with pre-loaded patterns ?
- Analysing the test scene images showed that some of the actual screws were never detected by the base model
- That means they never ended up in a memorised pattern
- Since the model performance is evaluated on a labelled images.
- The learning proposer will never have a chance to learn and propose those locations.
- This is most likely a limitation of the test data, because the test scenes are generated from same base scenes by applying random rotation, translation and light/contrast change.
- The issue may be less severe if those were real images for different instances of the same device.

ANALYSIS – LEARNING PROPOSER GAIN



ANALYSIS – NEGATIVE GAIN

- Is it possible that the 2-phase model decreases the performance of the base model?
- Consider the following scenario:
- Base Model detects 3 true positives, 1 false positive and dismisses 6 (false negatives)

  TP=3, FP=1, FN=6, Precision = TP/(TP + FP) = 0.75 , Recall = TP/(TP+FN) = 0.33,

  Base Model F1-score = 2*Precision*Recall / (Precision + Recall) = 0.458
- Now consider that proposer matches a wrong pattern and proposes 100 incorrect ROIs
- Out of those the Detector incorrectly classifies 10 ROIs as positives (false positives)

  TP=3, FP=1+10=11, FN=6

  Precision=TP/(TP + FP) = 0.21, Recall = 0.33

  2-Phase model F1 score = 2 * Precision * Recall / (Precision + Recall) = 0.26

  Gain = (2-phase model F1-score / base model F1-score) – 1 = (0.26 / 0.458) – 1 = -0.43

ANALYSIS – NEGATIVE GAIN

- How can we prevent the negative gain ?
- In the previous example, the proposer proposes an incorrect pattern with many ROIs
- The 2-phase model works best when the proposer proposes all of the locations that have been discarded by the base model (false negatives)
- If we know the base model recall and precision we can estimate the expected number of false negative (which should be approximately equal to the number of proposed ROIs).

  Expected FN = TP(1/Recall - 1) = (Precision x P) (1/Recall - 1)

- In previous example we have Base Model:

  P = TP + FP = 4, precision = 0.75, recall = 0.33

  Expected FN = (Precision x P) (1/Recall - 1) = 6 << number of proposed ROIs (100)

CONCLUSION

- The test results confirmed the project expectation
- In all test scenarios the 2-phase model outperformed the base model
- We have identified worse case scenario and proposed a mitigation
- Future works could explore using different proposer implementations, such as using Generative AI
- As well as combine scene images from multiple cameras
- Improving the quality of the Hough transformation extractor or replacing it with better (higher quality) implementation is another approach to improve the overall performance