Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 98 (2021) 666–671

www.elsevier.com/locate/procedia

28th CIRP Conference on Life Cycle Engineering

# Screw detection for disassembly of electronic waste using reasoning and re-training of a deep learning model

Gwendolyn Foo[a]*, Sami Kara[a], Maurice Pagnucco[b]

[a]*Sustainable Manufacturing and Life Cycle Engineering Research Group, School of Mechanical and Manufacturing Engineering, University of New South Wales, Sydney 2052, Australia*

[b]*School of Computer Science and Engineering, University of New South Wales, Sydney 2052, Australia*

* Corresponding author. Tel.: +61-2-9385-5698; *E-mail address:* gwendolyn.foo@student.unsw.edu.au

**Abstract**

Growing populations, increasing standards of living, and reliance on advancing technologies are resulting in an emerging abundance of electronic waste. Automating disassembly of these electronic products is a vital part of improving end-of-life product treatment where hazardous chemicals and materials are present. Furthermore, non-destructive disassembly is ideal to preserve the embodied energy in components from manufacturing. This requires the removal, and hence detection, of fasteners like screws in a disassembly environment. Crosshead screws are a common fastener type used in LCD monitors. This paper proposes a method of screw detection for disassembly and presents results of detection of crosshead screws on components of various models of LCD monitors in a disassembly cell. The system first applies gamma correction to standardize brightness—or luminance— of images. Generic knowledge about screw features in the form of a deep learning model is then used to visually detects screws. The results are analyzed against common screw location combinations on product components in order to logically reason about possible undetected screws that are also likely to exist. Finally, true negative detections are collected as new training data and the deep learning model is re-trained on these missed detections to improve performance; tailoring and adapting to the environment of the specific disassembly cell.



*Keywords:* disassembly automation; computer vision; screw detection; ontology; logical reasoning; LCD monitors;

## 1. Background

It is ideal for the disassembly process of end-of-life product treatment to be automated to reduce human exposure and risk to hazardous materials. Currently, commercial robotics and automation struggle to handle uncertainties and variations present in waste electrical and electronic equipment (WEEE). In particular, robotic systems need to be able to recognize components and fasteners and their locations in order to remove them. Many vision and object recognition techniques have been explored for this purpose.

A screwdriver assembly developed by Jin, *et al.* [1] included a ring shaped light source and industrial intelligent camera to detect and remove screws from LCD displays with 0.2mm accuracy. However, this requires the system to know approximate locations of screws for the robotic arm to inspect. Pattern recognition, template matching, Douglas-Peucker's algorithm and progressive probabilistic Hough Transform have been used for edge detection and region detection in [2]. This system was able to detect the battery cover, screws, and wires on a CD drive. Autonomous disassembly of car wheels was conducted by detecting contours and features of nuts with a stereo vision camera [3]. Two Xiaomi Yi cameras were implemented for stereo vision, creating a 3D point cloud of an object, but resulted in models with insufficient detail [4]. Another photogrammetry method involved obtaining multiple camera images and used multi-view stereo and Visual Structure for Motion algorithms [5]. This proved to be able to capture


10.1016/j.procir.2021.01.172

small features but was sensitive to having sufficient lighting and the 3D model was highly time consuming to reconstruct.

ResNet deep learning network architecture was utilized for part identification based on shapes and contours [6]. In [7], various deep neural network classifiers were assessed for screw detection and an optimal integrated classifier was developed. Harris corner detection, HSV analysis, and the use of a depth sensor detected screws including rusty screws in [8]. Another paper demonstrated screw detection in electric vehicle batteries using a Haar Cascade classifier [9] but the method was inflexible to rotated or angled visual instances of screws. However, these works have all been designed in a specific environment (with certain lighting conditions and camera equipment) and cannot handle uncertainties or learn further during application if implemented in a different environment. Furthermore, any missed screws that fail to be detected are final. Therefore, we propose a method for screw detection which can be adapted to the application's environment to improve results. Screws missed by the object detection model also still have the potential to be identified by visual reasoning.

## 2. System overview

The physical system consists of a Nikon D5100 DSLR camera equipped with a 35mm lens attachment mounted above and facing downwards onto the workstation where LCD monitors are mounted. The camera is connected to a computer via USB cable. The computer also runs the screw detection program, written in the Python programming language. The program reads the image taken by the camera and utilizes OpenCV 2 version 4.1.2 [10], Tensorflow version 1.14 [11] and OwlReady2 API [12] libraries to execute pre-processing, object detection and reasoning, respectively. The ontology for reasoning was created in Protégé version 5.5.0 [13], equipped with an add-on Pellet reasoner [14] plug-in.

The screw detection method presented involves three steps which are executed automatically by the program: pre-processing, screw detection via a deep learning model, and visual reasoning. An additional step is proposed to the user to tailor the deep learning model to the environment (lighting, camera angle, camera resolution, etc.) of the disassembly application. That is, images of screws which are missed by the program during application are collected and used to re-train the deep learning model. This re-training step can be undertaken at any time, as seen fit by the user but it is ideal to collect at least a hundred missed screw samples before re-training.

## 3. Screw detection

### *3.1. Pre-processing*

Four image pre-processing techniques were briefly explored using the OpenCV 2 image processing library [15]: (1) standard histogram equalization; (2) Contrast Limited Adaptive Histogram Equalization (CLAHE); (3) brightness and contrast adjustment; and (4) gamma correction.

Histogram equalization adjusts the contrast of an image according to its histogram, by stretching pixel values across a wider range. CLAHE works similarly but divides the image into small blocks and equalizes the histogram for each block individually, rather than the entire image globally. This allows areas of the image that have very bright pixels to be adjusted differently to areas that have very dark pixels. Histogram equalization and CLAHE functions in the OpenCV 2 library can only be applied to greyscale images. Gamma correction corrects the brightness, or luminance, of an image using a non-linear transformation defined by equation (1), such that when $\gamma < 1$, the brightness of dark areas of the image is increased more than areas of the image which are already bright [15]. Adjustment of brightness and contrast is self-explanatory.

$$f(x) = \left(\frac{x}{255}\right)^{\gamma} \times 255 \quad (1)$$

Some variation in each functions' parameters was explored, to demonstrate the general effects of each pre-processing method on the images and screw features and to ensure that they were within acceptable range for the given scenarios. The CLAHE function was implemented with a contrast limiting threshold of 2.0 and using blocks from dividing the image into an 8x8 grid. Contrast and brightness adjustments used parameters gain $\alpha$=2.2 and brightness $\beta$=50 respectively. The gamma correction factor was set to $\gamma$=0.4. Standard histogram equalization does not require any parameters.

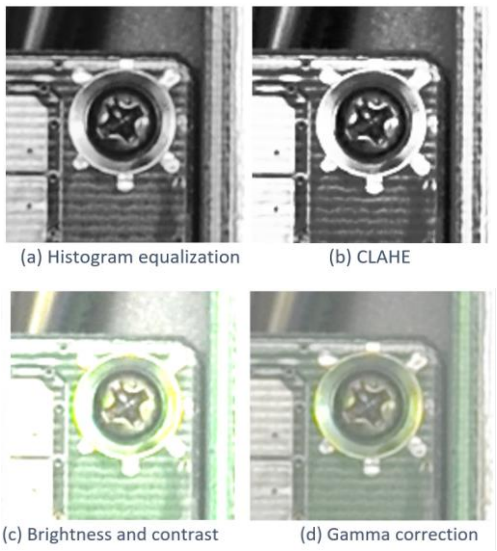


Fig. 1. Comparison of investigated image processing techniques.

The techniques were applied to images obtained from the camera on the disassembly cell. Histogram equalization and CLAHE work by enhancing details in the image. This resulted in slightly more frequent true positive detections than the other two techniques but also in more false positive detections as other non-screw features became enhanced so much that they were mistaken for screw features (Fig. 1a, b). Fig. 1c shows that the brightness and contrast adjustment method washed out images of shiny, smooth, metal components and intensified the darkness of deep grooves such as cross head features. Gamma correction in Fig. 1d appeared to reduce contrast in images and subsequently also muted important screw features. Based on these observations, brightness and contrast adjustment has been

implemented in this work. Intensifying cross head features and washing out other less prevalent features is expected to improve screw detection, particularly after the model is re-trained on application-obtained image data.

Each pre-processing technique produced similar results with object detection for screws prior to re-training. This reflects the consistency of the trained deep learning model to detect screws in various lighting and contrast conditions. Although pre-processing alone cannot improve screw detection results of an already trained deep learning model, re-training on pre-processed images of screws can make important features easier for the deep learning model to pick up on and recognize later.

### *3.2. Deep learning*

Initial training was done over 16 hours on 356 images of components from LCD monitors and other electronic products (including back covers, PCBs and carriers), and images obtained from Google image searches using keywords such as "screws" and "screw head". This provided the training set with image data of a variety of crosshead screws with different sizes, colors and textures: under different lighting conditions, with differing camera image resolutions, and at different angles and orientations. The labelImg graphical image annotation tool was used to manually label 1496 bounding boxes around clearly identifiable screws in these images [16].

A model pre-trained on the COCO dataset (*fastener_rcnn_inception_v2_coco*) [17] with the Faster R-CNN with Inception version 2 architecture is used for transfer learning. The training configuration with parameters used can be found at [18]. Transfer learning, which uses existing models' lower level layer parameters obtained from generic training images such as the COCO dataset [19], only needs to determine parameters for the final top layers of a CNN architecture using the customized, user-provided training images of the particular object(s) to be detected. This makes transfer learning require less time, training data, and computational power, while being sufficient in achieving object detection.

### *3.3. Visual reasoning*

The term *ontology* is used in this paper as defined in computer science, systems engineering and information science: *a representation of concepts within a domain and the structure of relationships between them* [20]. Generic knowledge about common products, components, fasteners, and disassembly tools and actions (in the form of classes and individuals) which are interconnected by relationships (object properties) and possess characteristics (data properties), was built into an ontology using Protégé version 5.5.0 opensource software [13]. The ontology class hierarchy is shown in Fig. 2a.

Subclasses relevant to screw detection include *Location*, *CrossHeadScrewFeature*, *Component*, *Product* and *Fastener*; and have been expanded to aid with understanding of their descriptions. Individuals are used as situational existences of the concept of which it is a subclass. For example, if there is an LCD monitor on the worktable which exists and which the disassembly system becomes aware of, an individual is created and stored as a subclass of *LCDmonitor* in the ontology. This individual represents only the specific LCD monitor on the worktable at that time, and all of the unique characteristics which it encompasses, even those that the system is not yet aware— the model, the orientation which it sits on the worktable, and the condition it is in; including dents, scratches, missing parts, etc. Semantic Web Rule Language (SWRL) rules are defined to allow the in-built reasoner to infer more complex relationships between individuals using these classes and properties. A SWRL rule consists of an antecedent (body) and consequent (head). If all conditions in the antecedent are true, then all conditions in the consequent must also hold true.

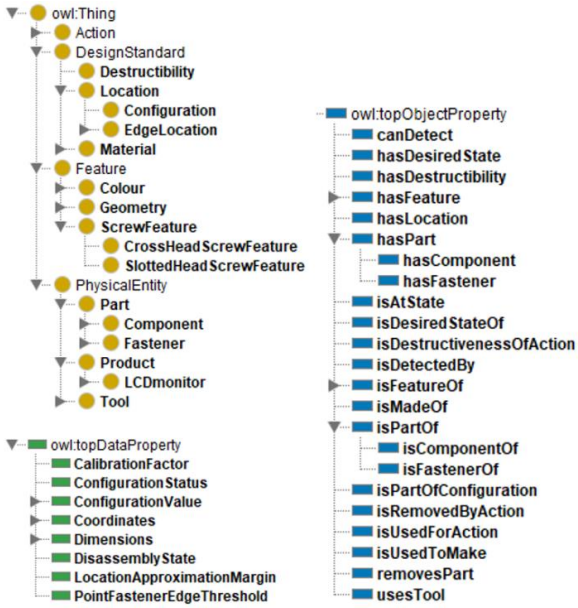


Fig. 2. Top left: (a) ontology class hierarchy, bottom left: (b) data properties, and right: (c) object properties.

Individuals of class *Component* or *Fastener* hold relational object properties (Fig. 2b) to descriptor individuals of a particular geometry or common feature. These feature individuals in turn hold Coordinates and Dimensions data properties (Fig. 2c) which are sets of numerical values describing location and size. *Fastener* individuals are also related to their corresponding *Component* via the object property *isFastenerOf*. The *Location* subclass *EdgeLocation* consists of four individuals shown in Fig. 3. Furthermore, the subclass of *EdgeLocation*, *CornerLocation*, includes four more individuals. The class *CornerLocation* is defined as a subclass of *EdgeLocation* as it holds the same properties but has additional SWRL rule requirements. The SWRL rules infer that a fastener individual is at an edge of the component if the *Coordinates* data property of its feature individual is located at a distance less than a predefined threshold value from the respective edge of the component. The left edge location point fastener example SWRL rule is provided in Algorithm I. The individual is also inferred to have a *CornerLocation* object property relation if it is related with two valid *EdgeLocation* individuals. For example, if a fastener is determined to be near to the top edge and left edge of a component, then it is also at the top left corner (Algorithm II).

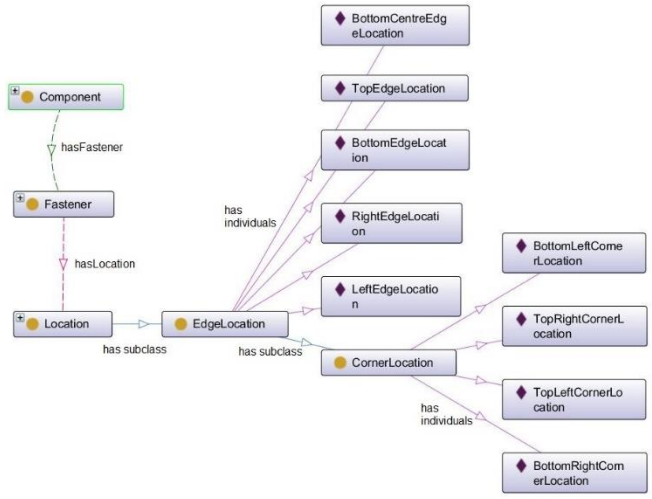


Fig. 3. Extract of disassembly ontology showing relationships between fastener class, location class, and their individuals.

Algorithm I. SWRL rule for left edge location point fastener.

```
PointFastener(?f) ^ Component(?c) ^ hasFastener(?c,
?f) ^ Coordinate_x(?f, ?fx) ^ Rectangle(?r) ^
isGeometryOf(?r, ?c) ^ Coordinate_x(?r, ?cx) ^
PointFastenerEdgeThreshold(DisassemblySystem,
?edgethresh) ^ swrlb:subtract(?diff, ?fx, ?cx) ^
swrlb:lessThan(?diff, ?edgethresh) ->
hasLocation(?f, LeftEdgeLocation)
```

Algorithm II. SWRL rule for top left corner location point fastener.

```
PointFastener(?f) ^ hasLocation(?f, TopEdgeLocation)
^ hasLocation(?f, LeftEdgeLocation) ->
hasLocation(?f, TopLeftCornerLocation)
```

The *Configuration* class represents an arrangement of locations which is common among electronic products and in manufacturing. Through analysis of manually disassembling LCD monitors, two common screw location configurations were found. Screws are often either configured in groups of four in the shape of a square, or in groups of three evenly spaced along the bottom edge of the component. As such, there are two individuals in the configuration class: *Configuration_4point* and *Configuration_3bottompoints*. SWRL rules are used to assign the object property *hasConfiguration* with one of these *Configuration* individuals to *Fastener* individuals of which that configuration applies. If there is one missing fastener which would otherwise make up a particular configuration, the reasoner infers that the screw detection model may have incorrectly made a true negative detection and adds a fastener at the expected location.

Point fasteners which are not identified as belonging to a complete configuration set are are analyzed by the reasoner to infer for any incomplete configurations with missing points.

## 4. Results and discussion

The testing procedure for the proposed screw detection method collected results from 6 scenarios for comparison (Table 1). The first 3 scenarios are run using the deep learning model initially trained on non-application specific screw images. The missed detections of screws are collected from Scenario 2, creating a new data set of application- and environment-specific screw images of adjusted brightness and contrast. 109 new screw labels were obtained from 24 images tested. Re-training was done using this data over 4.5 hours, starting from the last step of the initially trained model. These images were not used again for testing in Scenarios 4 to 6 because it is assumed that these disassembly instances have already occurred and the effect on images from new disassembly instances should be tested. Therefore, 16 new images, containing 67 instances of screws to be detected, were tested on all Scenarios 1 to 6 to evaluate the results from the various combinations of pre-processing, visual reasoning, and re-training the deep learning model. Testing images contained LCD monitors at differing orientations but always from a top angle looking down on the back cover.

Table 1. Testing scenarios.

| Scenario | Deep learning model screw detection | | Pre-processing | Visual Reasoning |
|---|---|---|---|---|
| 1 | ✓ | | | |
| 2 | ✓ | | ✓ | |
| 3 | ✓ | | ✓ | ✓ |
| 4 | ✓ | (re-trained) | | |
| 5 | ✓ | (re-trained) | ✓ | |
| 6 | ✓ | (re-trained) | ✓ | ✓ |

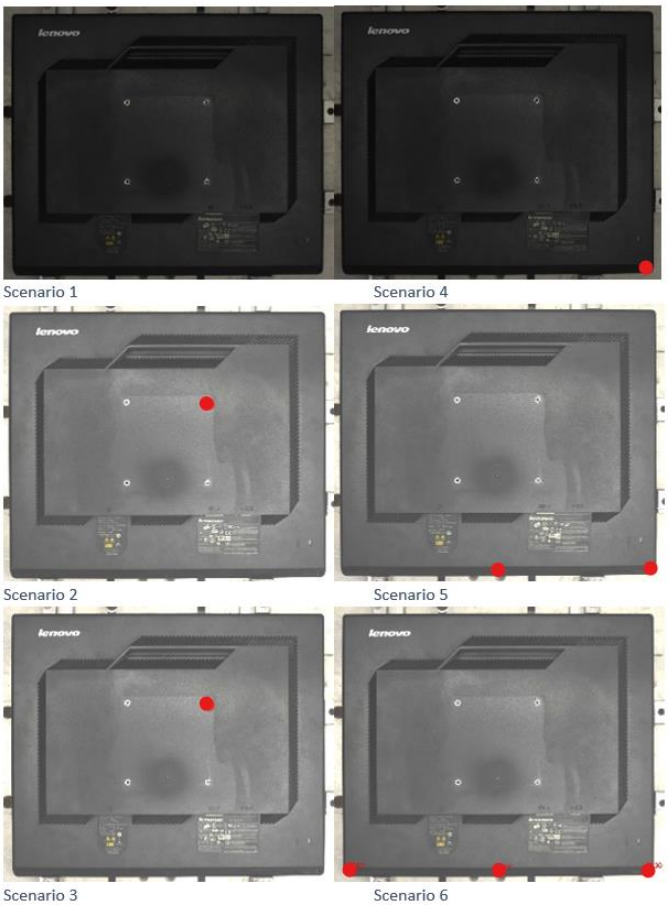


Fig. 4. Results of screw detection Scenarios 1 to 6 on an image from the disassembly cell.

A typical set of results from each scenario on one test image of an LCD back cover is presented in Fig. 4. The back cover

has 3 black screws equally spaced along the bottom edge of the cover, which are normally particularly challenging to detect by computer vision due to their color, size, location, and angle combination. This difficulty is evident in Scenario 1 where no screws are detected. No other features are incorrectly detected as screws either. Note that the 4 points in the middle of the LCD back cover are not screws but only empty screw holes where the LCD monitor stand was detached from. Therefore, the detection of these features as screws are false positive detections. After applying pre-processing, a false positive detection can be seen in Scenarios 2 and 3. This can be attributed to the fact that the initial training of the screw detection deep learning model was trained on images without the use of pre-processing.

In Scenario 5, some but not all screws are correctly detected since the deep learning model has now been re-trained on image data including images of screws with adjusted brightness and contrast. Finally, the ability of visual reasoning is demonstrated in Scenario 6. Where deep learning was not able to detect the screw on the bottom left of the back cover, the reasoner was able to consult the ontology and predict the location of the missed screw as shown.

For each scenario, the number of true positives (TP), false positives (FP), and false negatives (FP) are recorded along with the pixel coordinates of each detected screw. Precision, recall, and location accuracy are calculated and evaluated as performance measures from this data.

### *4.1. Accuracy*

Accuracy is calculated as per equation (2) where TN = 0 due to negative detections of screws not being explicitly made (anything not detected as screws are simply assumed to be non-screws). The results, which show a continual improvement from just 22% accuracy to a maximum of 78% accuracy in Scenario 6, are presented in Table 2.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (2)$$

Table 2. Accuracy results.

| Scenario 1 | Scenario 2 | Scenario 3 | Scenario 4 | Scenario 5 | Scenario 6 |
|---|---|---|---|---|---|
| 22% | 25% | 27% | 54% | 69% | 78% |

### *4.2. Precision and recall*

Precision is defined as the percentage of detections that are actually screws. Recall is defined as the percentage of screws that are successfully detected. These terms are used in line with definitions from Vongbunyong [21], and are expressed in equations (3) and (4) respectively.

$$Precision = \frac{TP}{TP+FP} \quad (3)$$

$$Recall = \frac{TP}{TP+FN} \quad (4)$$

A plot of calculated total precision and total recall for each scenario tested is shown in Fig. 5. As expected, using only the initially trained deep learning screw detection model in Scenario 1 resulted in the lowest total precision and total recall of 46.5% and 29.9% respectively. Both performance measures improved slightly with the addition of a brightness and contrast pre-processing technique in Scenario 2, and again with the addition of visual reasoning to infer about missed screw detections in Scenario 3. Furthermore, precision and recall rates significantly improved once the object detection deep learning model was re-trained on image data obtained from the application in Scenarios 4 to 6. Highest precision and recall rates of 91.8% and 83.6% respectively, were obtained in Scenario 6 by utilizing a combination of all three steps after retraining: pre-processing, object detection, and visual reasoning. This result is a significant improvement of 97.4% in precision and 180% in recall, from that of Scenario 1.

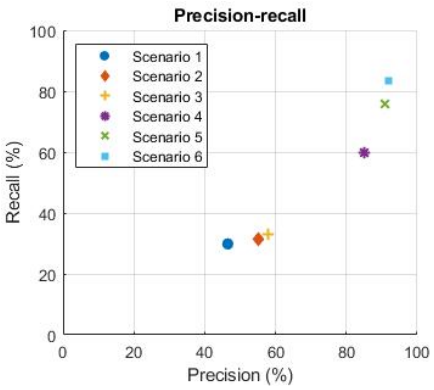


Fig. 5. Plot of total precision against total recall of each scenario tested.

### *4.3. Location accuracy*

Location accuracy, which is calculated according to equation (5), was measured by comparing how close the center-point of the detection $C_{detect}$ by the algorithm is to the actual center-point of the screw $C_{actual}$ in the image in pixels. $C_{actual}$ was obtained manually by selecting the pixel at the center of the screw head.

$$Location\ accuracy\ (mm) = |\boldsymbol{C_{detect}} - \boldsymbol{C_{actual}}| \times f \quad (5)$$

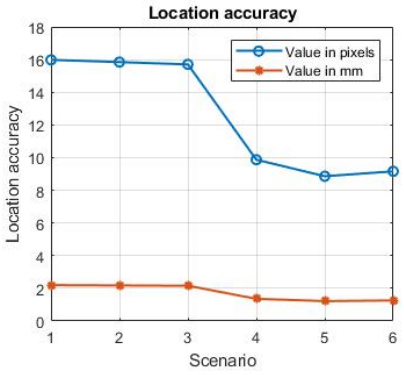


Fig. 6. Plot of average location accuracy for each scenario tested.

This is converted to a millimeter value based on a calibration factor $f$ of the camera image to real life measurements. A similar trend in improvement to that of precision and recall, can

also be observed in location accuracy (Fig. 6). That is, accuracy improved slightly from Scenarios 1 to 3 and then significantly between Scenarios 3 and 4 due to re-training. Finally, Scenario 6 achieved, on average, a location accuracy to 9.16 pixels, or 1.26 mm. This is a significant improvement of 42.7% from 15.98 pixels (2.19 mm) in Scenario 1.

## 5. Conclusion

Crosshead screws are a common fastener type used in products and the detection of them by computers can aid in automating disassembly for more sustainable EoL treatment. By re-training the model on environment-specific images obtained from the disassembly cell in application, combined with the knowledge of common fastener configurations to predict missed detections, screw detection is significantly improved. Furthermore, the model is able to detect screw heads with variation in rotation due to the training data being of screw heads in different 2-dimensional orientations which naturally occurs in WEEE. The orientation of the LCD monitor has little to no effect on the detections for this same reason. Nonetheless, images of LCD monitors in differing orientations were included in testing to ensure this.

The method is also able to handle small variations in angles of the screw head images that due to its location in relation to the overhead camera. That is, screws applied in a parallel direction to the camera's line of sight, so that the screw head is mostly facing the camera, is covered. Some screws may even be detectable at a sloping angle to the camera, but the extent of this is dependent on the screws which are identified for re-training. The method is not able to detect screws which have been applied on the sides of the LCD monitor since the screw head feature is not visible to the overhead camera, but these are minimal and other disassembly actions are being designed to address this.

Future work involves the implementation of this proposed method to a disassembly cell for application. Furthermore, the versatility of the proposed method could be tested for other screw types or fastener types prominent in disassembly.

## Acknowledgements

We would like to thank Jing Ying Gao, Dr. Yang Song, and Sebastian Blankemeyer for their advice and expertise in computer vision and deep learning methods.

## References

[1] W. Jin, S. Han, J. Fei, Z. Shi, W. Chen, W. Bao, J. Yang, W. Gong, J. Su, Y. Lv, D. Zhu, T. Zimei, J. Dou, and Q. Qin, "Design of platform for removing screws from LCD display shields," in LIDAR Imaging Detection and Target Recognition 2017, Changchun, China, 2017, vol. 10605, doi: 10.1117/12.2287626.

[2] P. Gil, J. Pomares, S. vT P. C. Diaz, F. Candelas, and F. Torres, "Flexible multi-sensorial system for automatic disassembly using cooperative robots," International Journal of Computer Integrated Manufacturing, vol. 20, no. 8, pp. 757–772, 2007, doi: 10.1080/09511920601143169.

[3] U. Büker, S. Drüe, N. Götze, G. Hartmann, B. Kalkreuter, R. Stemmer, and R. Trapp, "Vision-based control of an autonomous disassembly station," Robotics and Autonomous Systems, vol. 35, no. 3–4, pp. 179–189, 2001, doi: 10.1016/S0921-8890(01)00121-X.

[4] L. Jurjević and M. Gašparović, "3D Data Acquisition Based on Opencv for Close-Range Photogrammetry Applications," in The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Gottingen, Germany, 2017, vol. XLII-1/W1, pp. 377–382, doi: http://dx.doi.org.wwwproxy1.library.unsw.edu.au/10.5194/isprs-archives-XLII-1-W1-377-2017 [Online]. Available: http://search.proquest.com/docview/1985563327/abstract/959A9F2FF9E140CDPQ/1. [Accessed: 05-Jun-2020]

[5] M. U. R. Siddiqi, W. L. Ijomah, G. I. Dobie, M. Hafeez, S. Gareth Pierce, W. Ion, C. Mineo, and C. N. MacLeod, "Low cost three-dimensional virtual model construction for remanufacturing industry," Journal of Remanufacturing, 2018, doi: 10.1007/s13243-018-0059-5.

[6] J. Krüger, J. Lehr, M. Schlüter, and N. Bischoff, "Deep learning for part identification based on inherent features," CIRP Annals, vol. 68, no. 1, pp. 9–12, Jan. 2019, doi: 10.1016/j.cirp.2019.04.095. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0007850619301246. [Accessed: 11-Aug-2020]

[7] E. Yildiz and F. Wörgötter, "DCNN-Based Screw Detection for Automated Disassembly Processes," in 2019 15th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS), 2019, pp. 187–192.

[8] M. Bdiwi, A. Rashid, and M. Putz, "Autonomous disassembly of electric vehicle motors based on robot cognition," in 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 2500–2505.

[9] K. Wegener, W. H. Chen, F. Dietrich, K. Dröder, and S. Kara, "Robot Assisted Disassembly for the Recycling of Electric Vehicle Batteries," Procedia CIRP, vol. 29, pp. 716–721, 2015, doi: 10.1016/j.procir.2015.02.051.

[10] "OpenCV." [Online]. Available: https://opencv.org/. [Accessed: 10-Aug-2020]

[11] "TensorFlow," TensorFlow. [Online]. Available: https://www.tensorflow.org/. [Accessed: 10-Aug-2020]

[12] J.-B. Lamy, "Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies," Artificial intelligence in medicine, vol. 80, pp. 11–28, 2017.

[13] "Protégé." [Online]. Available: https://protege.stanford.edu/. [Accessed: 10-Aug-2020]

[14] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," Journal of Web Semantics, vol. 5, no. 2, pp. 51–53, Jun. 2007, doi: 10.1016/j.websem.2007.03.004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570826807000169. [Accessed: 08-Sep-2020]

[15] "OpenCV Tutorials." [Online]. Available: https://docs.opencv.org/3.4/d9/df8/tutorial_root.html

[16] T. T. Lin, labelImg. 2020 [Online]. Available: https://github.com/tzutalin/labelImg. [Accessed: 10-Aug-2020]

[17] "TensorFlow 1 Detection Model Zoo," GitHub. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md. [Accessed: 18-Nov-2020]

[18] "Faster R-CNN with Inception v2, configured for Oxford-IIIT Pets Dataset," GitHub. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/samples/configs/faster_rcnn_inception_v2_pets.config. [Accessed: 18-Nov-2020]

[19] "COCO - Common Objects in Context." [Online]. Available: https://cocodataset.org/#home. [Accessed: 08-Sep-2020]

[20] D. E. Forbes, P. Wongthongtham, C. Terblanche, and U. Pakdeetrakulwong, "Ontology Engineering," in Studies in Systems, Decision and Control, Springer International, 2018, pp. 27–40 [Online]. Available: https://dx.doi.org/10.1007/978-3-319-65012-8_3

[21] S. Vongbunyong, "Applications of cognitive robotics in disassembly of products," Ph.D. Thesis, University of New South Wales, Sydney, 2013.