Domain randomization using deep neural networks for estimating positions of bolts

Ezra Ameperosa
Air Force Research Laboratory,
Materials and Manufacturing Directorate (AFRL/RX)
Dayton, Ohio 45433
Email: ezra.ameperosa@us.af.mil

Pranav A. Bhounsule
Dept. of Mechanical and Industrial Engineering,
University of Illinois at Chicago
842 W. Taylor St., Chicago, IL 60607, USA
Corresponding Author Email: pranav@uic.edu

ABSTRACT

Current manual practices of replacing bolts on structures are time-consuming and costly, especially because of numerous bolts. Thus, an automated method that can visually detect and localize bolt positions would be highly beneficial. We demonstrate the use of deep neural networks using domain randomization for detecting and localizing bolts on a workpiece. In contrast to previous approaches that require training on real images, the use of domain randomization enables all training in simulation. The key idea is to create a wide variety of computer-generated synthetic images by varying the texture, color, camera position and orientation, distractor objects, and noise, and train the neural network on these images such that the neural network is robust to scene variability and hence provides accurate results when deployed on real images. Using domain randomization, we train two neural networks, a faster regional convolutional neural network for detecting the bolt and placing a bounding box, and a regression convolutional neural network for estimating the x- and y-position of the bolts relative to the coordinates fixed to the workpiece. Our results indicate that in the best case we can detect bolts with 85% *accuracy and can predict* 75% *of bolts within* 1.27 *cm accuracy. The novelty of this work is in using domain randomization to detect and localize: (1) multiples of a single object, and (2) small-sized objects* (0.6 *cm* $\times$ 2.5 *cm)*

1 INTRODUCTION

Fasteners such as bolts form an integral part of many structures (e.g., airplanes, cars, ships, bridges, machinery) and undergo maintenance cycles that involve either their tightening or replacement. Current practice involves visually detecting the bolts followed by manual tightening or measuring the coordinates using odometry (e.g., coordinate measuring machine, camera) and feeding these positions into the robot. Since there may be hundreds of such bolts, the manual approach is both time-consuming and costly. An automated method that can visually detect and localize bolt position would be highly beneficial. In this paper, we demonstrate the use of domain randomization—a technique in deep learning that enables the simulation-to-real transfer of learned neural networks with no training on real images—to automate detecting and localizing the position of multiple bolts in a workpiece.

There have been a few semi-automatic/automatic approaches on applications such as freight trains, satellites, and armored vehicles. In freight train application, cameras were installed alongside the track to capture images of bolts [1]. Then they trained a support vector machine classifier and combined with a rotate-and-slide window method to localize the three-bolt regions in the wheel followed by identification of the stereotypical hexagonal shape of the bolt region. In satellite application, ultrasonics were embedded in the structure [2]. Then, acoustic waves were sent and noting the time of flight it was possible to extract possible damaged locations. In this way, they identified loose bolts with an accuracy of 3 cm. In military applications, semi-automatic methods such as using augmented reality to help soldiers to install and remove fasteners inside armored vehicles have been used [3].

A problem similar to bolt localization is that of bolt-hole localization. Previous work in this area has used the circular geometry of the hole and used a circular Hough transform based feature extraction [4, 5] or template-based image matching [6] for localization. This process requires additional camera-workpiece calibration and image processing (e.g., segmentation), and a laser finder to detect the depth of the workpiece.

Furniture assembly robots such as the Ikeabot rely on the motion capture system for localization of the workpiece and then CAD models for the actual assembly [7]. However, the portability of a motion capture system and lack of accurate CAD models is a challenge in most bolt replacement applications. Another quick method is manual training of the robot to disassemble the assembled piece. Then assuming that the external conditions are unaltered, reversing the robot commands allows easy assembly [8]. While this technique is good for assembly-line manufacturing (i.e., assembly in bulk), it is not efficient for maintenance operations where only a few bolts might need replacement.

Traditional image processing techniques involve some variant of template matching. Templates of the objects to be detected and localized are generated either using point clouds or using CAD models. These are generated offline and stored in memory. At run-time, point clouds of the actual object to be localized are obtained from a camera. Then, features are extracted from the point cloud (e.g., using Viewpoint Feature Histogram (VFH) [9]) and matched with the offline database. This method requires color and depth information (e.g., using either an RGB camera and range finder or RGB-D camera such as Microsoft Kinect [10]). It is also possible to use multiple RGB cameras to reconstruct the point cloud [11].

Convolutional neural networks provide an alternate method for localization. This involves training one or multiple neural networks on either real or computer-generated images. For best performance, one needs a substantially large dataset.

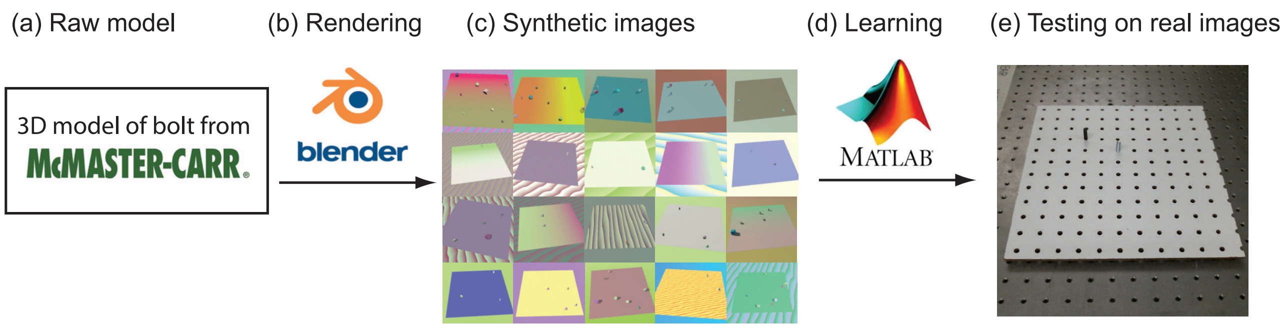


Fig. 1. **Work flow for the training and testing:** (a) the CAD model of the bolt is downloaded from McMaster Carr repository, (b) rendering is done using Blender, (c) a large number of synthetic images are generated, (d) training and validation of the convolutional neural networks are done in MATLAB, and (e) experimental image data is used for testing. Note that experimental images are not used for training.

However, creating a large dataset of real images is costly as it requires recording data and manual labeling the data. Hence, it is preferable to use computer-generated data for training the network. However, when identification is done on real images the learnt network leads to poor performance. This is the reality-gap in robotics and arises because of the difference between computer-generated images and real images. One can tackle this issue using an approach called domain adaptation and involves additional training of the neural network on real images [12, 13]. This vital step improves performance but comes at the cost of additional work needed to record and label experimental data. Recently, Tobin et al. [14] have demonstrated an alternate approach called domain randomization that uses simulated data for training without any calibration on real images. The key idea is to train the neural network on synthetically generated images with a wide variety of textures, colors, light conditions, camera positions and orientations, noise, etc. The expectation is that with enough variability in the synthetic data, the real word will appear just as an instance of the synthetic images providing high precision detection and localization, thus enabling the sim-to-real transfer. One of the big advantage of using a deep neural network is that the identification and localization is independent of the external factors (e.g., lighting, camera angle, color, texture) as one has to evaluate the network in real-time while traditional image processing needs further post-processing (e.g., feature extraction, template matching).

One can combine domain randomization and domain adaptation to realize new performance measures. A study showed that after domain randomization is used to tune the neural network, it requires 50 times less real-data for domain adaptation [16]. Although this efficiency is probably application dependent. Another study showed that tuning on synthetic data followed by real data is better than tuning purely on synthetic data [17]. Some application of domain randomization has been pick and place among cluttered objects [14], robotic grasping [18], and classification of objects [19].

Dynamic randomization is yet another complimentary technique that seeks to add variability in the simulated dynamics

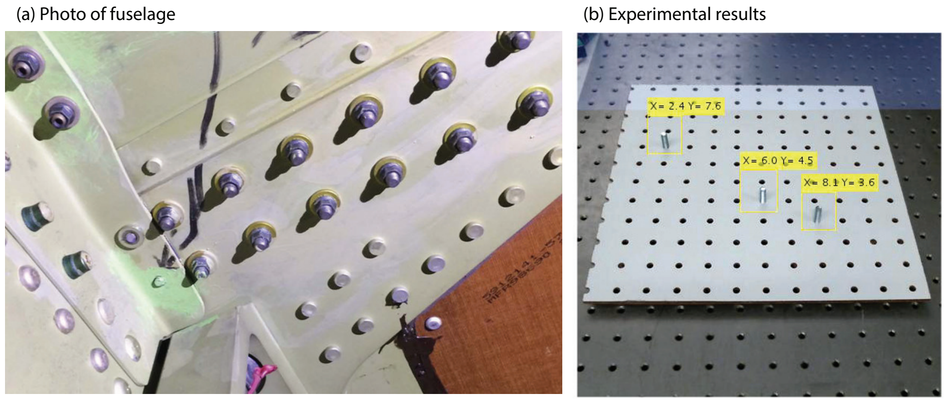


Fig. 2. **Example application and our experimental results:** (a) A cracked aircraft fuselage has numerous bolts that need to be removed for maintenance operations [15]. (b) Our experimental results show identification and localization of three bolts on a workpiece. We obtain the results by training a neural network entirely in simulation. That is, no re-training is done on the real images.

(e.g., vary the mass, damping, friction, etc.) followed by learning/control. Such controllers are robust to model uncertainty and transfer to the hardware with almost no hand-tuning. Some example of dynamic randomization involving robotic arm pushing an object [20], quadrupedal bounding [21], and humanoid movement control [22].

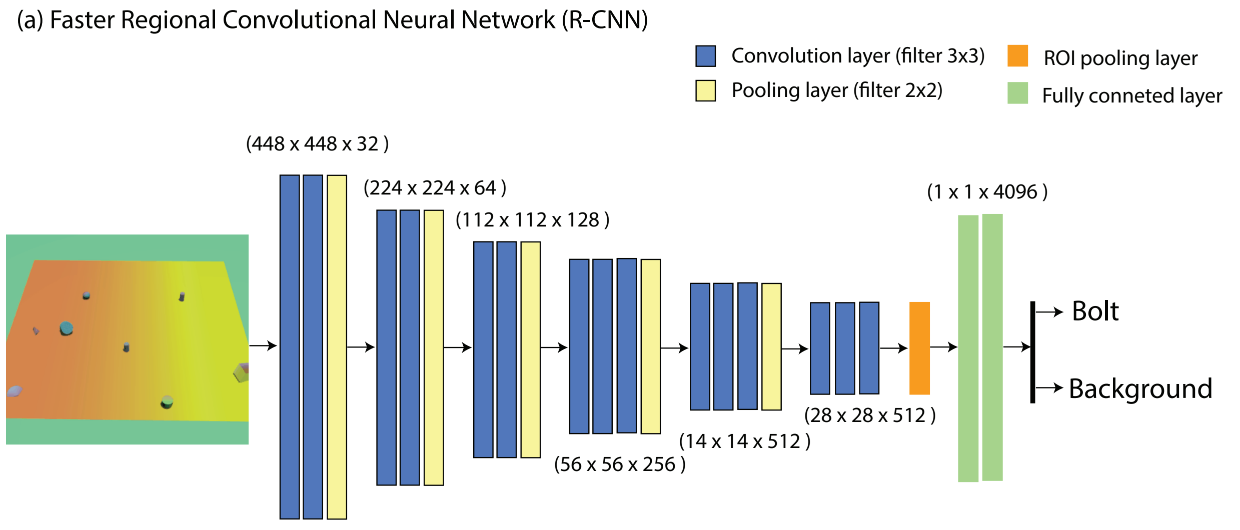
Convolutional neural networks have achieved wide usage in structural health monitoring and inspection. Some of these applications include crack detection in nuclear power plant components [23], LED chip defect inspection [24], monitoring faults on rotating machinery [25], detection of fabric defects [26], solder joints defects [27], defects in composite laminate beams [28]. With increasing computational power, especially parallel processing architectures and further developments in designing the convolutional neural network, we expect the use of CNNs for application to grow at a fast pace.

In this paper, we apply domain randomization for detecting and localizing bolts on a workpiece. An earlier version of this paper appeared in ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE) [29]. We adapt from the work by Tobin et al. [14] but is novel in the following ways: (1) we detect multiple objects and locate their positions, and (2) the object of interest, bolts are significantly smaller (0.6×2.5 cm). In particular, we use two neural networks; the first network detects multiples of the object from the world and the second one localizes the position of the bolt. We organize the rest of the paper as follows. We present the methods including metrics for evaluation in Sec. 2, the results in Sec. 3, the discussion in Sec. 4, and conclusions, including proposed future work, in Sec. 5.

2 METHODS

2.1 Overall approach

We depict our overall approach in Fig. 1. As shown in Fig. 1 (a), we obtain 3-D models of the bolts from McMaster-Carr's online repository [30]. We use the open-source software Blender [31] to render scenes with one or two bolts as shown in Fig. 1(b). In addition, we vary the lighting, color, texture, workpiece angle, and also add distractor objects. This leads to the generation of synthetic image data as shown in Fig. 1 (c). Then, as shown in Fig. 1 (d), we use the synthetic data set for training and validation using convolutional neural networks in MATLAB. Finally, in Fig. 1 (e) we test on experimental



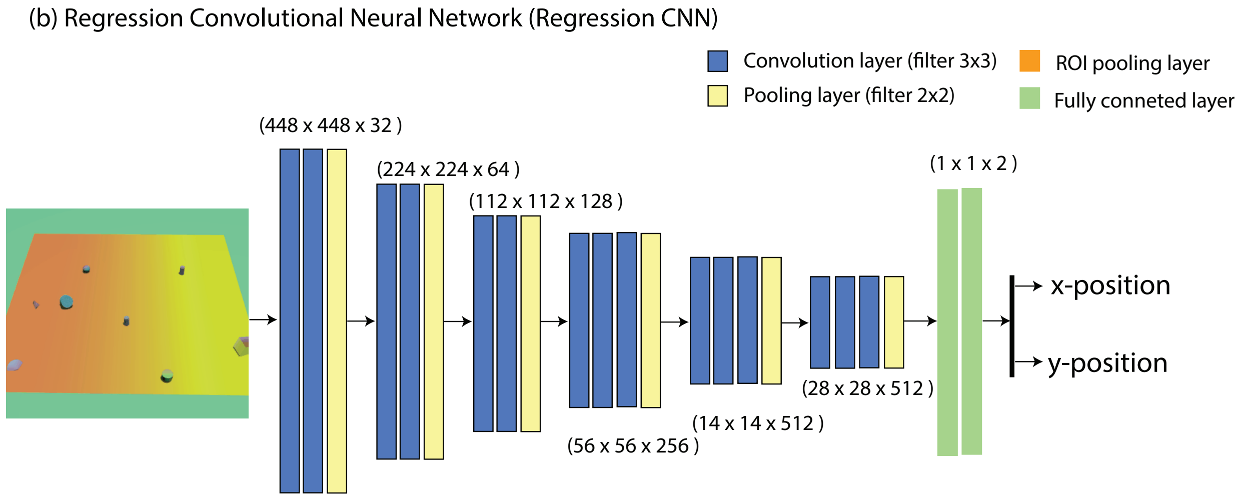


Fig. 3. **The two neural networks used for detecting and localizing multiple bolts:** (a) A Faster Regional Convolutional Neural Network (Faster R-CNN) is used to classify multiple bolts, and (b) A Regression Convolutional Neural Network (regression CNN) is used to localize the position of the bolt.

images which we obtain from a red-green-blue (RGB) camera. Note that we do not train the convolutional neural networks on the experimental images. This method known as domain randomization uses only synthetic data with a substantial amount of variabilities for training followed by evaluation on experimental images [14]. Figure 2 shows an example result of our algorithm detecting the three bolts on the workpiece and then estimating their position.

2.2 Computer generated images for training

As mentioned earlier, we obtain 3-D models of the bolts from McMaster-Carr's online 3-D models [30]. Then we use the open-source software Blender [31] to generate synthetic images for training. We place the workpiece at the center of the scene and we set its size to 30.5 $\times$ 30.5 cm. We set a table beneath the workpiece that to occupy space in the camera frame not covered by the workpiece. To create data for training, we create scenes resembling our physical system and include enough variability by randomizing the following factors: (1) the camera position and orientation; (2) lighting position and

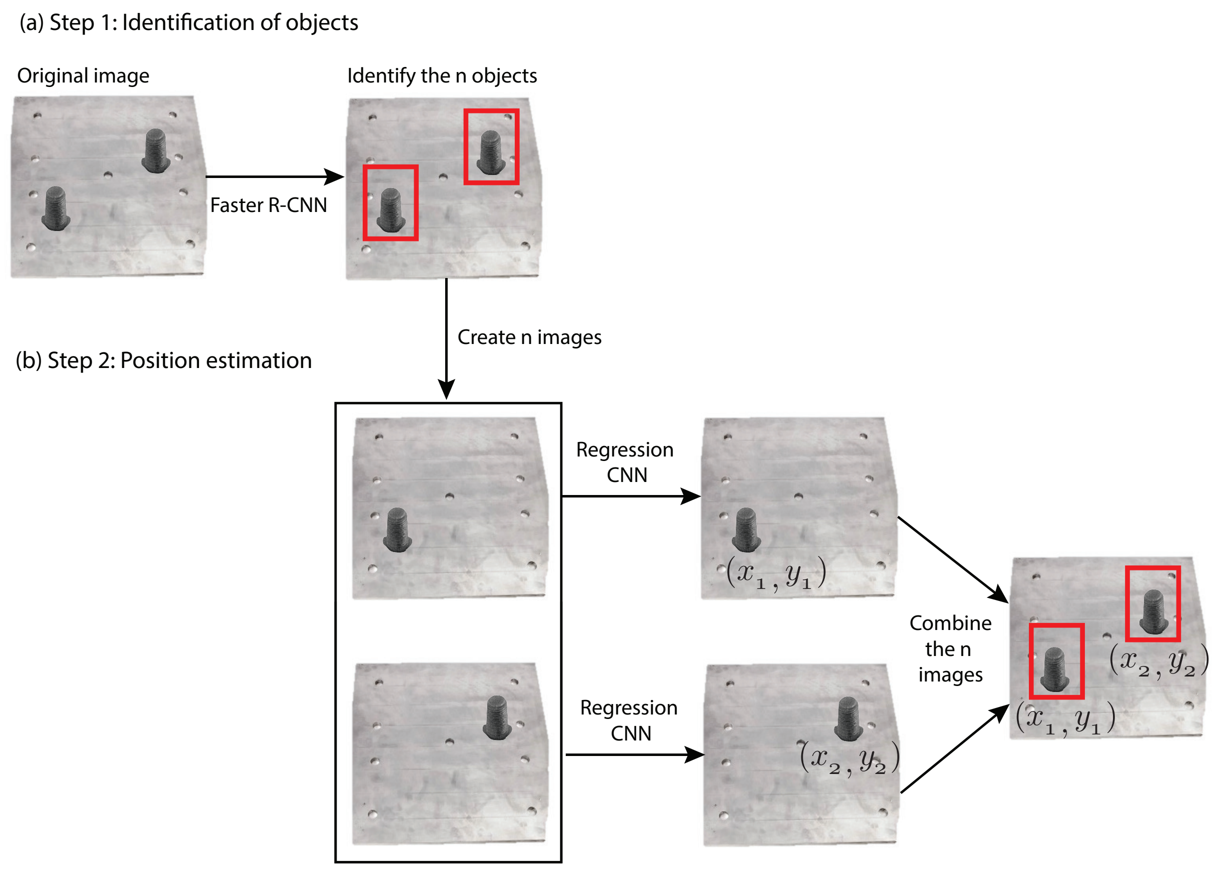


Fig. 4. **Overview of our sequential method for object detection and position estimation:** (a) Step 1 involves identifying the n objects of interest (here $n = 2$) and creating a bounding box for each of them using Faster R-CNN, and (b) step 2 involves creating n images, each with a single object followed estimation of the position of each object using Regression CNN.

properties; (3) number, shape, and position of distractor objects; (4) number and position of the bolts (either 1 or 2 bolts) on the workpiece; and (5) color and texture of the table, workpiece, bolts, and distractor objects. We generate scenes with these randomized features using Blender. We render the scenes at an image resolution of $448 \times 448 \times 3$. We create $75,000$ images in Blender (see Fig 1 (c) for some examples), with $50,000$ images containing only one bolt and $25,000$ images containing two bolts on the workpiece. Note that since we generate all these images synthetically, we know the position of the bolts precisely and there is no additional work in estimating the bolt position for labeling. We give more details about the factors that add variability to the synthetic data set in the following section.

2.2.1 Camera

In conventional vision-based localization, we need to calibrate the camera with respect to the workpiece. This involves mapping the camera coordinates and axis to the workpiece coordinates and axis respectively and the image size with that of the workpiece size. However, in our method, we bypass this step by varying the camera position and orientation as part of our training data. The rationale is that the neural network can automatically find the mapping between the image and workpiece when given enough variability in the data. Another advantage of the step is that we can change the camera position and orientation within certain limits during testing without affecting the results.

We use a single camera in Blender to capture the synthetic images and randomize the camera's position with respect to the center of the workpiece. We position the camera $[-7.6, 7.6]$ cm from the center in the x-direction, $[-30.5, -45.7]$ cm away in the y-direction, and $[30.5, 45.7]$ cm above in the z-direction (see Fig. 2). After randomizing the position, we aim the camera to the center of the workpiece. We also randomize the camera's Field of View in the range $[35^\circ, 40^\circ]$.

2.2.2 Lighting

We create two lighting sources; the first lighting source is the key lighting and the second source is the backlighting. We give the key lighting a yellow tint while we give the backlighting a light blue tint to mimic lighting in our actual testing environment. The backlighting is also used to soften shadows made by the key lighting. We randomize the key lighting's intensity and the backlighting is approximately half of the intensity of the key lighting with some added variance.

We position the key light above the workpiece occupying a semi-spherical space with a radius of 2.13 m from the center of the workpiece. We position the backlighting 180° about the center of the workpiece from the key lighting. Both light sources point at the center of the workpiece. We also include a 10° variation to generate the training data.

2.2.3 Distractor objects

We create and position the distractor objects in the training data. They act to prevent the neural network from overfitting the synthetic data. Without the use of these distractor objects, the neural network would be vulnerable to false-positive detections of random objects and any features not captured in training.

We create four shapes in Blender as distractor objects: cubes, spheres, cylinders, and cones. The distractor objects are of the same size as the bolts. We assign from 0 to 8 distractors in each scene randomly picked from the 4 shapes mentioned above. We place the distractors in arbitrary positions on the workspace.

2.2.4 Bolts

We position one or two bolts anywhere on the workpiece but excluding the workpiece edges. Placing the bolt on the edge would leave the bolt shank jutting past the workpiece. This would be problematic since we measure the bolt's position from the center of the bolt shank. For each bolt, we setup the bounding boxes in the synthetic images using Blender and the Cartesian coordinates (x- and y-position) of each bolt with respect to an origin fixed at the bottom left corner of the workpiece. We use the bounding boxes and x- and y-position as the ground truth for training the neural network.

2.2.5 Color and texture

We set up three textures for the bolt, distractor objects, table, and workpiece: solid color texture, marble texture, and a gradient texture. For each of the objects in the scene, we randomly set a texture to the objects and randomize the colors of the texture. We also add the possibility for objects to have 0% to 60% reflectivity.

2.3 Neural Networks

We use two neural networks: a Faster Regional Convolution Neural Network (Faster R-CNN) [32] for identifying objects and putting a bounding box around it and a regression Convolutional Neural Network (regression CNN) to estimate the x- and y-position of a single bolt. We base both these on convolutional neural networks (CNN) that use filters to identify and learn features without using too many parameters. The Faster R-CNN incorporates a regional proposal network (RPN) besides a CNN that helps to identify multiple objects and create a bounding box around them. For regression CNN, we add a regression layer to enable localization of the object. We use two instances of the VGG16 architecture with pre-trained weights from ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [33] for these networks. We also modify the networks to accept images of resolution $448 \times 448 \times 3$ rather than $224 \times 224 \times 3$. We use a higher resolution than the default lower resolution to enable the use of a high-resolution camera because of the small size of the bolts.

2.3.1 Training Faster R-CNN

We use VGG16 with pre-trained weights from ILSVRC. We modify the network to accept 448×448 images as mentioned earlier but also add two convolutional layers with ReLu activations following each layer. The convolutional layers have 32 filters with a filter size of 3×3. We add padding size of one and use a stride of one. We add a max-pooling layer after the second convolution layer with a pool size of two, zero padding, and a stride of two. We add these new layers to the beginning of the VGG16 architecture. We change the last fully connected layer to a two-neuron layer and then change the Softmax and classification layer to have an output label of either a bolt or background. We show our network in Fig. 3 (a).

We use the function *trainFasterRCNNObjectDetector* in MATLAB 2018a. We split the $75,000$ synthetic images (containing one or two bolts) into a training set of $67,500$ and validation set of $7,500$. We use the training set and the bounding boxes taken from Blender as input to the network. We then use stochastic gradient descent with momentum because MATLAB 2018a does not allow Adam optimization, which is the natural choice for Faster R-CNN. We use 0.9 for the momentum coefficient. We also add an L2 regularization with a coefficient of 10^{-4}. We use a mini-batch size of 256 and train for three epochs. The training took about 1 day using graphic card GTX1080 on a standard desktop computer (circa 2017).

2.3.2 Training regression CNN

The regression CNN is the VGG16 architecture, but we modify final layer to perform Mean Square error rather than Cross Entropy loss. The regression CNN shown in Fig. 3 (b) is almost identical to the Faster R-CNN with only one change. We replace the ROI pooling layer in Faster R-CNN (orange-colored block just before the last two fully connected layers shown using green blocks in (a)) with a pooling layer (yellow block just before the two fully connected layers shown using a green block in (b)). We use the fully connected layers for regression and they output the x- and y-position of the bolt.

We use the function *train* in MATLAB 2018a. We split the $50,000$ synthetic images that contain one bolt into a training set of $49,000$ and a validation set of $1,000$. We do not use the images with two bolts ($25,000$ images) as the neural network in MATLAB can only estimate the position of one bolt at a time. We use Adam optimization with an initial learning rate of 10^{-4}, and a drop rate factor of 0.1 for every four epochs. We use a mini-batch size of 32 and train for 9 epochs with no L2 regularization. The training took about 4 hours using graphic card GTX1080 on a standard desktop computer (circa 2017).

2.4 Sequential algorithm for testing on real images

Figure 4 shows the workflow used for testing the trained neural networks on real images. First, we pass the image through the Faster R-CNN to detect the bolts and put a bounding box on it as shown in (a). After we identify each of the n bolts, we create n images by blending out all but one bolt as shown in (b). We then pass each of the n images individually through the Regression CNN to estimate the x- and y-position of each bolt. Finally, we combine all the images into a single image by putting a bounding box and labeling the position of each bolt.

2.5 Metrics

2.5.1 Metrics for identification

Our metrics for evaluation of the Faster R-CNN for identification are the “Recall”, “Precision”, and “F$_1$ Score”. We define these next

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{1}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{2}$$

$$\text{F}_1\ \text{Score} = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} \tag{3}$$

Recall (Eqn. 1) is the ratio of correctly identified instances to all correct instances. It measures how well the detector can find all instances of the bolts. Precision (Eqn. 2) is the ratio of the correctly identified instances to the total number of positive detections. This measure indicates how many detections are actual bolts among all the instances where the detector thought it was a bolt. With precision and recall, we can better measure how well our detector performed in contrast to measuring just how many bolts it detected out of the total bolts. We use the precision and recall in finding the F$_1$ Score (Eqn. 3) at varying minimum acceptable confidence scores. The F$_1$ score is the harmonic mean of both recall and precision and is our performance score. An F$_1$ value close to 1 indicates a perfect recall and precision and gets worst as the value approaches 0. We use this metric to find the score for different confidence thresholds in detecting the bolt.

2.5.2 Metric for position estimation

The metric for evaluation of the regression CNN is the “Position error”, which we define as

$$\text{Position Error} = \sqrt{(x-\hat{x})^2 + (y-\hat{y})^2} \tag{4}$$

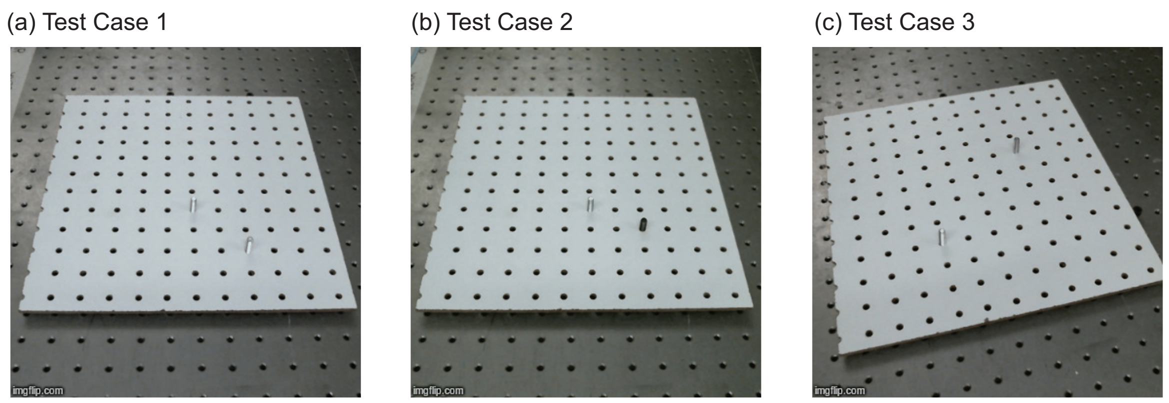


Fig. 5. **The three test sets:** (a) Test set 1: same color bolts, straight camera, (b) Test set 2: different color bolts, straight camera, (c) Test set 3: different color bolts, skew camera, different work piece colors.

where x and y are the true positions of the bolt, $\hat{x}$ and $\hat{y}$ are the position estimated by the regression CNN. Note that the origin for the measurement is the bottom left corner of the workpiece.

3 RESULTS

Besides the results given here, we provide a video showing the training and experimental verification and the blender/MATLAB code in Appendix A: Multimedia extension.

3.1 Experimental setup

Our workpiece is 30.5 $\times$ 30.5 cm wooden pegboard, which we place on an optical breadboard table. We create three test cases comprising 20 scenes. We show the test cases in Fig. 5. All test cases are similar in that they all have two bolts randomly placed on the pegboard but differ in the following ways. In test case 1, both bolts were stainless steel bolts. In test case 2, one bolt was a stainless steel bolt while the other was a black oxide bolt. In test case 3, we have a random combination of stainless steel, black oxide, and cadmium-plated bolt (a golden look). Also, for test case 3, we randomly chose either a white or a brown-colored pegboard (the workpiece). The camera position for all test cases was at an arbitrary distance and height but we position the camera straight ahead for test case 1 and test case 2 but it was 7.6 cm to the left for test case 3. We did not calibrate the camera at any point of the testing. This is a significant advantage of using domain randomization.

3.2 Results for identification

We plot the F_1 score in Fig. 6 for the three test cases for detection confidence bounds between 50% and 65%. On average, the F_1 scores for test cases 1, 2, and 3 are 0.75, 0.88, and 0.84 respectively. Note that F_1 scores are measures of false positives (see Eqn. 2) and false negative (see Eqn. 1). From these results we see that test case 2, which has bolts of different colors, has the highest F_1 scores and the test case 1, which has the same color bolts, has the lowest F_1 score. We observe that test case 1 and 3 both have about the same number of false positives, and they appear in the same area throughout the sets: the bottom left part of the images, and the mid-right side of the images. We also observe that the detector has only failed in detecting bolts on the right side of the workpiece. However, we did not investigate the cause for these observed patterns. But

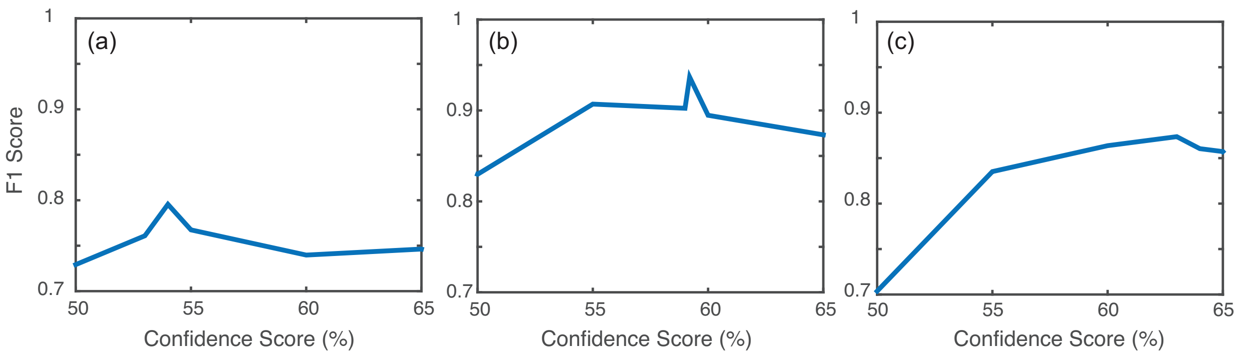


Fig. 6. **F1 score for the three test cases** (a) Test case 1, (b) Test case 2, (c) Test case 3.

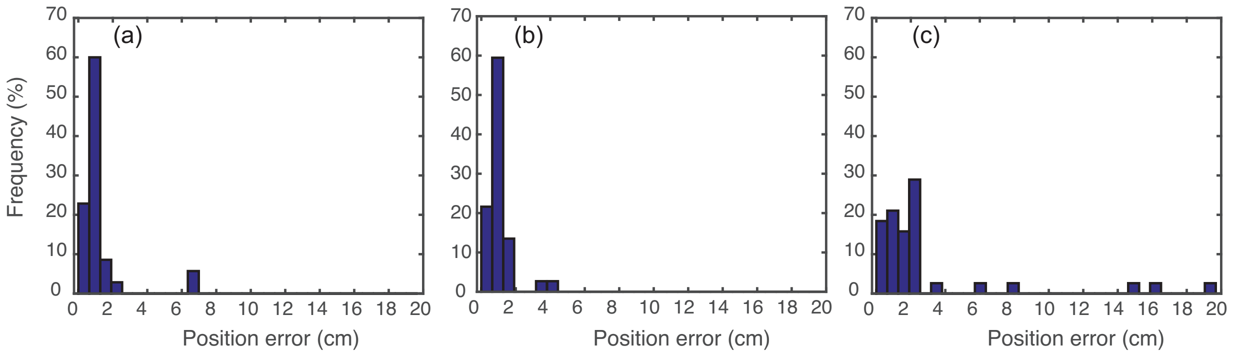


Fig. 7. **Histogram for the three test cases as a function of the position error:** (a) Test case 1, (b) Test case 2, (c) Test case 3.

we suspect that our domain randomization might not have covered certain combinations leading to poor performance in this part of the workpiece.

3.3 Results for position estimation

We show the results for accuracy of the regression CNN for position estimation in Fig. 7 and Table 1. The y-axis in the histogram shows the percentage with respect to the number of bolts correctly identified, which are 35, 37, and 38 for the three test cases (see row 1 in Tab. 1) and the x-axis shows the position error in cm (see Eqn. 4).

The results for test case 1 and test case 2 are almost similar as seen from Fig. 7 (a) (b) and Tab. 1 columns 3 and 4. The average errors for test case 1 and test case 2 were 1.22 cm and 1.07 cm respectively. The test case 2 results are marginally better than test case 1. Note that the only difference between the test cases is that test case 1 uses similar bolts (steel bolt) while test case 2 uses different bolts (a steel bolt and a black oxide bolt). The distinct colors in test case 2 leads to a better position estimation. The average error for test case 3 was 2.95 cm, about 3 times worse than test case 1 and test case 2. The main difference in test case 3 was the movement of the camera position to the right, thus showing that the neural network is more sensitive to this parameter. We also notice that substantial errors in Fig. 7 (c) in the range $4-20$ cm. This is because of substantially high variability in this data set, which we may have missed to represent adequately in our training dataset.

Table 1. Performance metrics for the three test sets to the two neural networks

Metric	Neural Network	Test Set 1	Test Set 2	Test Set 3
Bolts correctly identified	Faster R-CNN	35 out of 40	37 out of 40	38 out of 40
Average Error (cm)	Regression CNN	1.22	1.07	2.95
Standard Deviation (cm)	Regression CNN	1.35	0.74	4.39
Bolts with <1.27 cm or 0.5" error	Regression CNN	29 out of 35	30 out of 37	15 out of 38

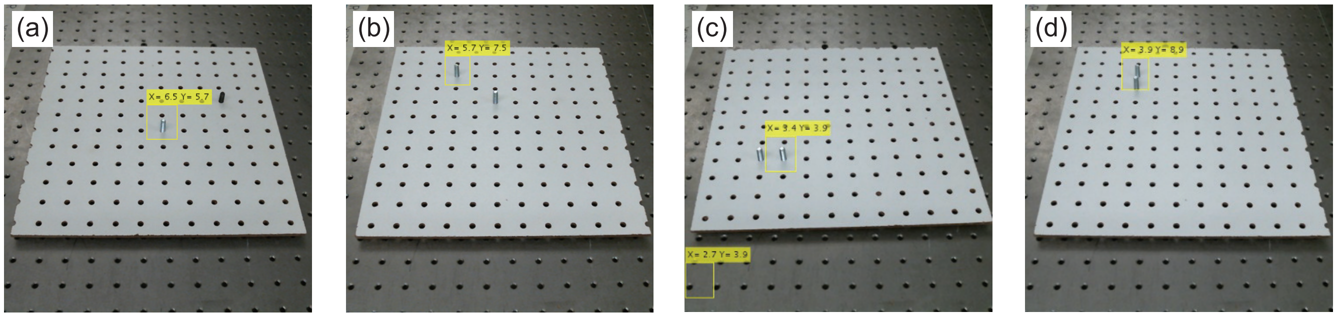


Fig. 8. **Limitations of our method:** (a) Only one bolt detected, (b) only one bolt detected but with incorrect position, (c) only one bolt detected due to close proximity of the bolts, (d) two bolts identified as one by the bounding box.

4 DISCUSSION

In this paper, we have demonstrated the use of domain randomization—a technique that relies only on training on simulation followed by testing on real hardware—to identify and estimate the position of multiple bolts on the workpiece. Our approach comprises using the Faster Regional Convolutional Neural Network (Faster R-CNN) for object identification and laying down bounding boxes around the bolts followed by using a Regression Convolutional Neural Network (Regression CNN) for estimating the position of the bolts. We can detect between 88% to 97% of the bolts with an average error between 1.07 cm to 2.95 cm depending on the difficulty of the task.

Our work is novel in two aspects: (1) detection and estimation of the position of multiple instances of the same object, and (2) detecting small objects, as each bolt is 0.6 cm in radius and 2.5 cm. We base our work on Tobin et al. [14], who used domain randomization to detect spam objects among a series of randomly chosen objects. In their case the objects were big (we estimate 5 cm or more) and they were only detecting a single object in the frame. Despite the smaller sized objects in our case, we could achieve similar position estimation accuracy, 1.5 cm. However, note that we only estimate the x-, y- position while Tobin et al. estimated the x-, y-, and z- position of the object.

A key aspect of our method is to use separate networks for identification (Faster R-CNN) and position estimation (Regression CNN). Although it takes more time to tune two networks independently, this has the advantage of being able to test and debug independently. For example, we can retrain the network which is performing poorly rather than training a single network. Another feature is that we used the networks sequentially; first, Faster R-CNN detects the number of bolts and second, regression CNN determines the position using the results from the Faster R-CNN output. The advantage of this sequential composition is that for training the regression CNN we only need to train on a single bolt, not multiple. However, a disadvantage of the sequential algorithm is that the results of the position estimation using Regression CNN depend on the results of the identification using Faster R-CNN. For example, in Fig. 8 (a) we could detect only one bolt and thus we could estimate the position of only one bolt. We show another example in Fig. 8 (b) where, although we detect one bolt, it has co-ordinates of the other bolt. We explain this as follows. With our method the algorithm would make two images, blurring out one bolt in one image then blurring out the other bolt in the second image, then send it through the position estimating network. However, because the detector only identified one bolt, it does not blur any bolts and sends the unedited images through the second network. Through this, we cannot control which bolt the second network looks at to estimate its position. Here, we erroneously assign the position of the undetected bolt to the detected bolt.

We have used $75,000$ images in total for training and validation. We arrived at $75,000$ by iterating between the number of images and the fidelity of the fit. We kept increasing the images till there was no appreciable change in the loss function for the complexity of the convolutional neural network. This result showed that the further increasing the number of images would have a negligible impact on the fit.

Our algorithm has a difficult time localizing the bolts when they are close to each other as shown in Fig. 8 (c-d). In (c), the proximity of the two bolts leads Faster RCNN to identify only one bolt while in (d) we identify two bolts as a single one as shown by the bounding box. We can eliminate this problem if we had a closer view of the bolts. Another way would be to limit the bounding box size. Yet another way would be to use a higher resolution image. One limitation of MATLAB 2018a was that we have reached the minimum ROI size for the bounding box and cannot reduce it further without possibly risking compromising the network. Ambient lighting is an important consideration during deployment of the system. Though we did not explicitly change the lighting in the room, we did our experiments throughout the day in natural lighting. Our results were consistent, showing that our system is robust to ambient lighting. However, it would be better to control the lighting to validate the robustness to lighting conditions.

We find that many of our limitations are to do with the neural network toolbox in MATLAB 2018a. For example, MATLAB's minimum bounding box size does not allow us to identify multiple closely placed bolts. Another issue is that MATLAB does not allow simultaneous identification and regression, and thus we had to use two different neural networks independently for training. One way to do this would be to change the RPN by adding more layers such that it does the bounding box regression loss to include a weighted x and y position regression loss. While MATLAB is continually improving its toolboxes TensorFlow is a more mature and robust system for the image processing using neural networks. In hindsight, at the current time, TensorFlow seems to be a better choice for deep learning of images than MATLAB. We chose MATLAB purely based on the availability of the neural network toolbox in our organization and the ease of using other toolboxes (e.g., computer vision for obtaining images from the camera).

5 CONCLUSION AND FUTURE WORK

This paper demonstrates that small objects such as bolts in structures may be detected and localized leveraging state-of-the-art neural network classifiers and regressors and trained only on synthetic data using domain randomization. We believe that accuracy of 1.5 cm achieved through this method on hardware provides sufficient precision for robot grippers to pick and place tiny objects such as bolts (typically of size 0.6×2.5 cm).

There are multiple future research directions:

1. Test the algorithm on additional bolts and conditions (e.g., distraction objects, lighting, camera position).
2. Combine the classification and regression networks into a single neural network and compare its performance with separate networks as done here.
3. Extend the regression to estimate the z-position and the orientation of the bolt on the workpiece.
4. Demonstrate robotic pick-and-place with objects localized using domain randomization.

References

[1] Li, C., Wei, Z., and Xing, J., 2016. "Online inspection system for the automatic detection of bolt defects on a freight train". *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit,* **230**(4), pp. 1213–1226.

[2] Reynolds, W. D., Doyle, D., and Arritt, B., 2010. "Active loose bolt detection in a complex satellite structure". In Health Monitoring of Structural and Biological Systems 2010, Vol. 7650, International Society for Optics and Photonics, p. 76500E.

[3] Henderson, S. J., and Feiner, S., 2009. "Evaluating the benefits of augmented reality for task localization in maintenance of an armored personnel carrier turret". In 2009 8th IEEE International Symposium on Mixed and Augmented Reality, IEEE, pp. 135–144.

[4] Baek, G. R., Mo, Y. H., Jeong, J. S., Park, J. M., and Lim, M. T., 2011. "Cognition system of bolt hole using template matching". In 28th International Symposium on Automation and Robotics in Construction, ISARC 2011, Citeseer.

[5] Nagarajan, P., Perumaal, S. S., and Yogameena, B., 2016. "Vision based pose estimation of multiple peg-in-hole for robotic assembly". In International Conference on Computer Vision, Graphics, and Image processing, Springer, pp. 50–62.

[6] Choe, Y., Lee, H.-C., Kim, Y.-J., Hong, D.-H., Park, S.-S., and Lim, M.-T., 2009. "Vision-based estimation of bolt-hole location using circular hough transform". In ICCAS-SICE, 2009, IEEE, pp. 4821–4826.

[7] Knepper, R. A., Layton, T., Romanishin, J., and Rus, D., 2013. "Ikeabot: An autonomous multi-robot coordinated furniture assembly system". In Robotics and Automation (ICRA), 2013 IEEE International Conference on, IEEE, pp. 855–862.

[8] Miller, J. M., and Hoffman, R. L., 1989. "Automatic assembly planning with fasteners". In Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on, IEEE, pp. 69–74.

[9] Rusu, R. B., Bradski, G., Thibaux, R., and Hsu, J., 2010. "Fast 3d recognition and pose using the viewpoint feature histogram". In Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, IEEE, pp. 2155–2162.

[10] Zhang, Z., 2012. "Microsoft kinect sensor and its effect". *IEEE multimedia,* **19**(2), pp. 4–10.

[11] Collet, A., and Srinivasa, S. S., 2010. "Efficient multi-view object recognition and full pose estimation". In Robotics

and Automation (ICRA), 2010 IEEE International Conference on, IEEE, pp. 2050–2055.

[12] Gopalan, R., Li, R., and Chellappa, R., 2011. "Domain adaptation for object recognition: An unsupervised approach". In Computer Vision (ICCV), 2011 IEEE International Conference on, IEEE, pp. 999–1006.

[13] Matasci, G., Tuia, D., and Kanevski, M., 2012. "Svm-based boosting of active learning strategies for efficient domain adaptation". *IEEE J. Sel. Topics. Appl. Earth Observ. Remote Sens.,* **5**(5), pp. 1335–1343.

[14] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P., 2017. "Domain randomization for transferring deep neural networks from simulation to the real world". In Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on, IEEE, pp. 23–30.

[15] Australian_Government, 2018. Airbus a380-842 horizontal stabiliser structure—bushing migrated. sdr 510020443. `http://www.flightsafetyaustralia.com/2015/03/17-november-2014-19-january-2015-3/`, July.

[16] Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., et al., 2018. "Using simulation and domain adaptation to improve efficiency of deep robotic grasping". In 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp. 4243–4250.

[17] Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., To, T., Cameracci, E., Boochoon, S., and Birchfield, S., 2018. "Training deep networks with synthetic data: Bridging the reality gap by domain randomization". In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 969–977.

[18] Tobin, J., Zaremba, W., and Abbeel, P., 2017. "Domain randomization and generative models for robotic grasping". *arXiv preprint arXiv:1710.06425*.

[19] Borrego, J., Dehban, A., Figueiredo, R., Moreno, P., Bernardino, A., and Santos-Victor, J., 2018. "Applying domain randomization to synthetic data for object category detection". *arXiv preprint arXiv:1807.09834*.

[20] Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P., 2018. "Sim-to-real transfer of robotic control with dynamics randomization". In 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp. 1–8.

[21] Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., and Vanhoucke, V., 2018. "Sim-to-real: Learning agile locomotion for quadruped robots". *arXiv preprint arXiv:1804.10332*.

[22] Mordatch, I., Lowrey, K., and Todorov, E., 2015. "Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids". In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp. 5307–5314.

[23] Chen, F.-C., and Jahanshahi, M. R., 2017. "Nb-cnn: deep learning-based crack detection using convolutional neural network and naïve bayes data fusion". *IEEE Transactions on Industrial Electronics,* **65**(5), pp. 4392–4400.

[24] Lin, H., Li, B., Wang, X., Shu, Y., and Niu, S., 2019. "Automated defect inspection of led chip using deep convolutional neural network". *Journal of Intelligent Manufacturing,* **30**(6), pp. 2525–2534.

[25] Ince, T., Kiranyaz, S., Eren, L., Askar, M., and Gabbouj, M., 2016. "Real-time motor fault detection by 1-d convolutional neural networks". *IEEE Transactions on Industrial Electronics,* **63**(11), pp. 7067–7075.

[26] Tsai, I.-S., and Hu, M.-C., 1996. “Automatic inspection of fabric defects using an artificial neural network technique”. *Textile Research Journal,* **66**(7), pp. 474–482.

[27] Ko, K. W., and Cho, H. S., 2000. “Solder joints inspection using a neural network and fuzzy rule-based classification method”. *IEEE Transactions on Electronics Packaging Manufacturing,* **23**(2), pp. 93–103.

[28] Zheng, S.-j., Li, Z.-q., and Wang, H.-t., 2011. “A genetic fuzzy radial basis function neural network for structural health monitoring of composite laminated beams”. *Expert Systems with Applications,* **38**(9), pp. 11837–11842.

[29] Ameperosa, E., and Bhounsule, P. A., 2019. “Domain randomization for detection and position estimation of multiples of a single object with applications to localizing bolts on structures”. In ASME 2019 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers.

[30] McMaster-Carr, 2018. Online catalog of mechanical, electrical, plumbing, and hardware parts. `https://www.mcmaster.com/`, July.

[31] Blender, 2018. Blender, a free and open-source 3d computer graphics software toolset. `https://www.blender.org/`, July.

[32] Ren, S., He, K., Girshick, R., and Sun, J., 2015. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In Advances in neural information processing systems, pp. 91–99.

[33] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al., 2015. “Imagenet large scale visual recognition challenge”. *International Journal of Computer Vision,* **115**(3), pp. 211–252.

Appendix A: Multimedia Extension

1. A video of the hardware prototype available on this YouTube link:
   `https://youtu.be/-DLU-bjDOhE`.
2. The MATLAB and Blender code with test examples are on github: `https://github.com/EzAme/DR2`.

List of Figures

List of Tables