ASME Journal of Computing and Information Science in Engineering
Online journal at:
https://asmedigitalcollection.asme.org/computingengineering

Transforming Hand-Drawn Sketches of Linkage Mechanisms Into Their Digital Representation

Anar Nurizada
Computer-Aided Design and Innovation Lab,
Department of Mechanical Engineering,
Stony Brook University,
Stony Brook, NY 11794-2300
e-mail: anar.nurizada@stonybrook.edu

Anurag Purwar[1]
Computer-Aided Design and Innovation Lab,
Department of Mechanical Engineering,
Stony Brook University,
Stony Brook, NY 11794-2300
e-mail: anurag.purwar@stonybrook.edu

This paper introduces a new method using deep neural networks for the interactive digital transformation and simulation of n-bar planar linkages, which consist of revolute and prismatic joints, based on hand-drawn sketches. Instead of relying solely on computer vision, our approach combines topological knowledge of linkage mechanisms with the outcomes of a convolutional deep neural network. This creates a framework for recognizing hand-drawn sketches. We generate a dataset of synthetic images that resemble hand-drawn sketches of linkage mechanisms. Next, we fine-tune a state-of-the-art deep neural network to detect discrete objects using building blocks that represent joints and links in various positions, sizes, and orientations within these sketches. We then conduct a topological analysis on the detected objects to construct a kinematic model of the sketched mechanisms. The results demonstrate the effectiveness of our algorithm in handling hand-drawn sketches and converting them into digital representations. This has practical implications for improving communication, analysis, organization, and classification of planar mechanisms.
[DOI: 10.1115/1.4064037]

Keywords: planar linkage mechanisms, simulation, machine learning, deep learning, object detection

1 Introduction

During the product design process, sketching plays a pivotal role in effectively conveying and visualizing ideas. Within engineering design teams, designers frequently create kinematic sketches of mechanisms to aid in the brainstorming process. The automated detection of critical components in linkage mechanisms, including pivot type and location, link dimensions, and interconnection patterns, has the potential to facilitate various tasks such as transferring a mechanism from a sketch to computer simulation software, digitally cataloging and classifying existing designs, and generating similar mechanisms for concept development.

This paper introduces a framework for the automated and real-time digital conversion of hand-drawn sketches depicting planar linkage mechanisms. The approach involves training a specialized deep convolutional neural network (CNN) using a synthetic database of linkage sketches. The trained CNN exhibits the capability to detect multiple objects within an image, providing bounding boxes and class probabilities for each detected object. These bounding boxes are subsequently adaptively resized in preparation for the subsequent topology analysis phase, which determines joint and link types, connections, link dimensions, and pivot locations. The final output is presented in the form of an adjacency matrix, as well as tables detailing links and joints. An overview of this approach is depicted in Fig. 1.

Recent years have seen numerous attempts to combine hand-drawn sketches with machine learning for the development of methods that enable automated detection of shapes and their features. Huang [1] utilized a CNN-based deep neural network to facilitate user interface (UI) design by sketching a target UI and then retrieving UI screenshots from large-scale datasets. Güçlütürk et al. [2] demonstrated the synthesis of photo-realistic human faces from sketches. Ellis et al. [3] trained a CNN to convert simple hand-drawn sketches into graphics programs written in LaTeX. Simo-Serra et al. [4] presented a novel technique to simplify sketch drawings by training a CNN to remove unnecessary lines in a sketch.

Moreover, there has been a growing interest in developing a machine learning method for synthesizing 3D models from 2D sketches. Oh et al. [5] devised a novel way of using deep generative design frameworks for creating different design options optimized for engineering performance. A similar work was proposed by Pu et al. [6], which lets a user sketch a 2D shape in the way engineers usually draw three views of 3D models—front, top, and side view—and get a 3D version of the shape from a pre-compiled database. Willis et al. [7] tackled the problem of learning based engineering sketch generation as a first step toward synthesis and composition of parametric computer aided design models. Para et al. [8] introduced SketchGen, which is a generative model based on a transformer architecture to address the heterogeneity problem by carefully designing a sequential language for the primitives and constraints. Kazi et al. [9] developed DreamSketch, a 3D design interface that combines the free-form and expressive qualities of sketching with the computational power of generative design algorithms. Murugappan et al. [10] convert sketches to online sketches and use existing stroke-based recognition techniques for further processing. The converted sketch can be edited, segmented, recognized, merged, solved for geometric constraints, beautified, and

[1]Corresponding author.

Manuscript received February 19, 2023; final manuscript received November 1, 2023; published online November 30, 2023. Assoc. Editor: Yan Wang.

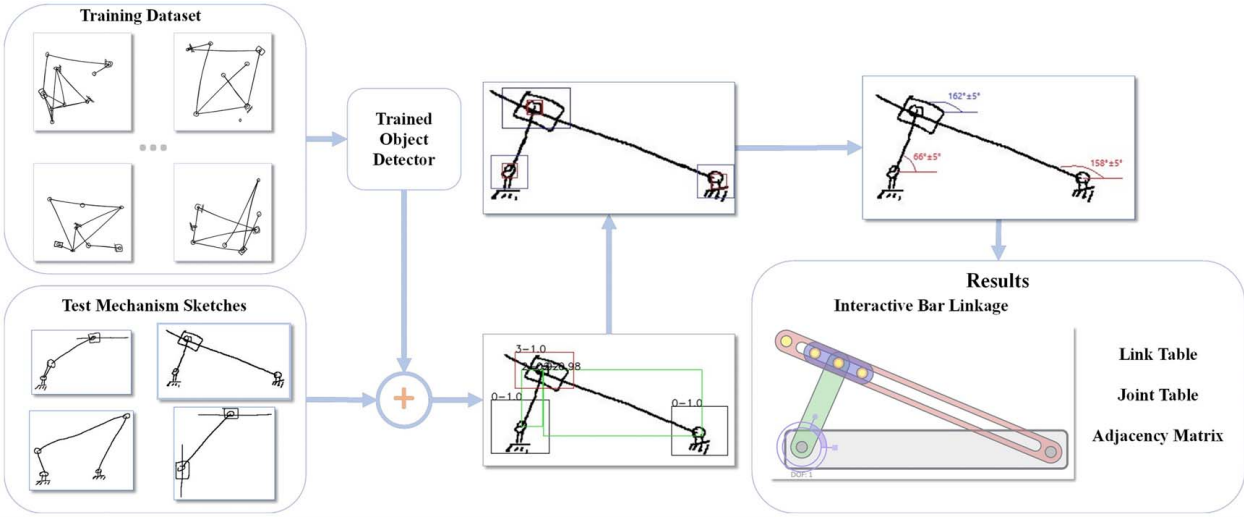

Fig. 1 Overview: a CNN object detector is trained with a set of synthetic hand-drawn sketches of mechanisms, which are not required to be valid mechanisms. During testing, users input a sketch of a bar linkage (one of the test mechanism sketches). The CNN outputs bounding boxes and labels around detected objects. The output bounding boxes are resized and connections between the joints and links are determined. If there are any prismatic joints present, we find links along which sliders move. Using the information about the location of joints and their connections with links, the linkage geometry is determined and presented in joint–link tables and adjacency matrix.

used as input for finite element analysis. Additionally, Murugappan and Ramani presented a program called FEAasy [11] for performing finite element analysis on sketches of structural models. Finally, Nie et al. [12] utilized a CNN to predict the stress fields in 2D linear elastic cantilevered structures.

The application of machine learning in kinematic synthesis has gained increasing attention in the recent years. Nobari et al. [13] introduced LINKS, a comprehensive dataset comprising a hundred million planar linkage mechanisms, created to support data-driven approaches in kinematic design. Deshpande and Purwar [14,15] introduced a novel approach for synthesizing defects-free, planar four-bar mechanisms by integrating pattern recognition, machine learning, and rigid body kinematics. They also proposed a deep generative model for concept generation of mechanisms [16]. More recently, Purwar and Chakraborty [17] published a position paper on deep learning design of robot mechanisms, which outlines potential avenues for future research in this area.

In this research, we combine kinematic analysis with machine learning to develop a framework for simulating sketch-based mechanisms. Our objective is to create an automated method for identifying different components of an *n*-bar linkage mechanism, such as links, free and fixed revolute (R) and prismatic (P) joints, from a sketch of the mechanism. This work offers an opportunity for mechanism design researchers to employ a well-labeled dataset of mechanisms for concept generation. Potentially, a computational tool could be provided to mechanism designers to sketch their ideas or an existing mechanism from texts or patents. Using the approach outlined in this study, the tool could produce a labeled mechanism suitable for supervisory machine learning techniques.

Eicholtz and Kara were the first one to present an approach for image-based recognition of planar mechanical linkages found in textbooks and hand-drawn sketches [18–20]. Their method incorporated a sliding window technique [21] alongside histograms of oriented gradients [22] and a pre-trained soft linear support vector machine [23] to identify potential joints within an input image. Upon the detection of joints, mechanical constraints, specifically mechanisms with one degree-of-freedom, were enforced, and all feasible connections between joints were determined using the nondominated sorting genetic algorithm (NSGA-II) [24]. The approach is effective for both textbook illustrations and sketches of bar mechanisms. Furthermore, it introduces the concept of the *user effort*, which informs users about the number of steps required to correct a sketched mechanism. A higher score indicates that fewer steps are needed to refine the sketched mechanism, while a lower score implies the opposite [20].

While this approach exhibited promise in identifying sketched mechanisms, it did not distinguish between fixed and free pin joints and was not suitable for prismatic joints lacking revolute joints overlaid on them, as seen in a slider-crank mechanism. Furthermore, the average correct detection rate for mechanisms was relatively low, and the work predates the availability of real-time object detectors, relying entirely on computer vision techniques.

In contrast, this paper proposes using a real-time object detector to detect joints and links, followed by topological analysis to find connections and geometric parameters of mechanisms. The contributions of this work are threefold: (1) presenting a methodology for creating a dataset of synthetic images of sketches with hand-drawn quality of planar linkages, including arbitrarily placed revolute and prismatic joints; (2) fine-tuning a real-time object detector to detect discrete building blocks of linkage mechanisms; and (3) utilizing topological knowledge of mechanisms to identify joint types, joint–link connections, and output geometric information for further processing in kinematic design and simulation tools like MotionGen [25,26]. By doing so, this work establishes a synergy between machine learning techniques and linkage analysis, leveraging existing knowledge from kinematics.

Rest of this paper is organized as follows. In Sec. 2, data preparation methodology for object detector training is presented. Section 3 is dedicated to the details of fine-tuning the detector with planar *n*-bar linkage sketches and Sec. 4 presents an algorithm for topology determination using detected bounding boxes. Section 5 presents accuracy results of running the object detector on different kinds of *n*-bar linkages before presenting conclusions.

2 Synthetic Image Generation of Sketches

Training a CNN-based object detector requires a large number of input images, which in our case would be a dataset of hand-drawn planar linkage sketches. Unfortunately, there are no such datasets available. One way of generating a dataset would be extracting images of mechanisms from the textbooks, patents, and internet or asking people to sketch bar mechanisms and upload them; however, this method would be very time-consuming and not

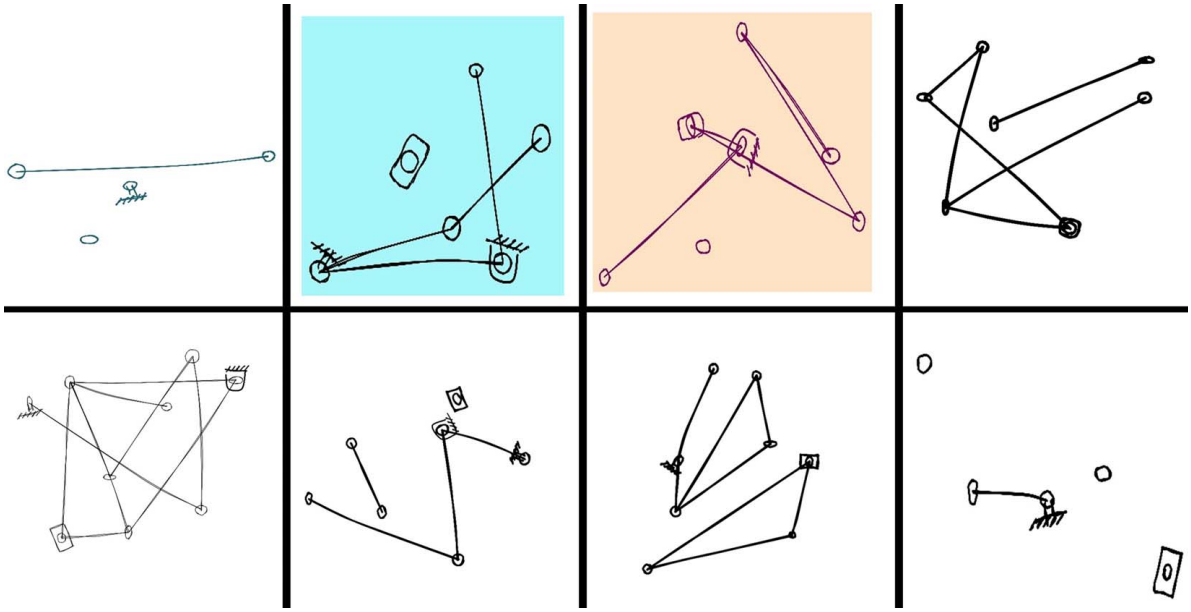

Fig. 2 Top row left to right: gray stroke on white background, gray background with black stroke, mix of gray background and stroke, black and white sketch with stroke 6. Bottom row left to right: stroke thickness 1, 3, 4, 5.

yield enough training images needed to retrain an object detector. Thus, in this paper, we introduce an automated method for generating images of n-bar linkages by writing a script, which uses Scribble library [27,28] for simulating hand-drawn shapes. Using our script, we generated 18,000 training, 2000 validation, and 6500 testing images in less than an hour. Training the object detector also requires generating a file for each training image which contains ground truth information about all objects present in an image. This would further increase manual dataset gathering time since each image would be manually annotated. Our script provides annotation automatically as well. In addition to that, one could update our script to include more ways or styles of drawing bar mechanisms. Images scraped from the texts, internet, or from student assignments were not used as part of this dataset.

The training dataset for fine-tuning the object detector consisted of black and white images depicting different parts of bar linkages, such as fixed and free revolute joints, links, and prismatic joints, some of which are connected. Each image had a different number of links, free, fixed, revolute, and prismatic joints. This work assumes that since most people use a pencil and a white paper to draw sketches, there is no need to train the detector on colored images. However, real-life images of sketches might be colored and backgrounds might not be perfectly white; e.g., if one uses a flash while taking an image of a sketch, it might result in the central part of an image being brighter than the other part. In order to ensure that our model performs well in those cases, we made additional testing datasets: (1) with backgrounds of different shades of gray, i.e., the stroke of the mechanism was black, whereas the background was filled with different shade of gray, (2) with gray-colored stroke on a white background, (3) a mix of different shades of gray background and gray stroke, and (4) different stroke thicknesses, ranging from thickness level of 1 (thinnest) to 6 (thickest). The detector was trained on black and white sketches with a thickness of 2. Thus, we compiled an additional 4500 testing images, bringing a total of our testing images to 6500. Our results showed that the color of a sketch and thickness levels of 1–3 do not make any difference; however, thickness of 4 and higher decreases the detection accuracy. However, these thickness levels result in the stroke being unrealistically thick; thus, we made another assumption that users will not input such sketches and there was no need to retrain the detector with such sketch images. An image from each dataset is shown in Fig. 2. The last assumption about the input sketch images in this work is that there is no textual information present in the image.

The dataset must cover as many ways of drawing a planar n-bar linkage as possible; i.e., different locations of fixed and free joints, their orientation, type, quantity, etc. At this stage, our goal was to create not well-formed mechanisms, such as feasible four-bar linkages, etc., but rather an ensemble of connections of joints and links that would cover a high variety of possible ways of drawing bar mechanisms as shown in the Training Dataset box in the Fig. 1. The dataset is accessible via a link.[2]

2.1 Mechanics of the Script. In each iteration or frame of the script, a fresh canvas measuring 600×600 pixels is initialized. The script proceeds to generate and draw ten ellipses on this canvas. These ellipses have randomly selected radii ranging from 10 to 30 pixels, and their positions are determined randomly within the canvas. To ensure that ellipses are not drawn too close to the canvas borders, their x and y coordinates are constrained within a range of 50–550 pixels.

To prevent overlapping, if the distance between any two randomly generated ellipses is less than 70 pixels, one of them is removed. This constraint results in varying the total number of joints drawn in each frame, ranging from five to ten, with an average of seven joints per frame.

Furthermore, every second ellipse is drawn as a fixed joint, while every third one is drawn as a slider joint. The rest are rendered as free joints. It is worth noting that the specific pixel values, such as the radius range of 10–30 pixels, were chosen empirically. This selection ensures that the ellipses maintain a realistic appearance, neither appearing as mere points (when the radius is too small) nor excessively large (when the radius is too large).

The next step of the program involved defining an $N \times N$ zero matrix, where N represents the total number of joints in a frame. Subsequently, the program assigned values of 1 to certain lower triangular entries of the zero matrix, following a Bernoulli distribution with a probability of 30%; that is, assigning 1 with a probability of 30%. While iterating through this matrix, the program established a link between two joints, denoted as i and j, if the corresponding

[2]https://www.kaggle.com/datasets/anarnurizada/n-bar-mechanisms

entry in the matrix was equal to 1. For instance, the following matrix was employed to create the links depicted in Fig. 3.

$$[T] = \begin{pmatrix} & J1 & J2 & J3 & J4 & J5 \\ J1 & 0 & 0 & 0 & 0 & 0 \\ J2 & 0 & 0 & 0 & 0 & 0 \\ J3 & 0 & 0 & 0 & 0 & 0 \\ J4 & 0 & 0 & 1 & 0 & 0 \\ J5 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Diagonal entries were kept zero, since a joint cannot be connected to itself. It is important to point out that not all joints were connected since this would result in an unrealistic sketch and mostly over constrained mechanisms. An excessive number of links would make it difficult to differentiate various elements in the drawing even for a human eye. This way the final dataset consisted of 18,000 training, 2000 validation, and 6500 testing images.

A sample generated image is shown in Fig. 3. It has a total of five joints, three of which are free revolute joints, one is a fixed revolute joint, and one is a prismatic joint. There is a total of one link connecting two random joints. On the other hand, Fig. 4 shows an unrealistic image example where almost all joints are connected. This image has eight joints and 14 links which are hard to differentiate even visually. In a case where there are ten joints per image, differentiating between links would become even harder.

Fine-tuning of the object detector on a custom dataset requires annotation for each training image, which provides information about all the detectable objects in the image. This annotation is of the following format: object's *class*, object's bounding box's center x value, object's bounding box's center y value, bounding box *width*, bounding box's *height*. Object class relates to the class of an object, which in our case is either a free, slider, or fixed joints, or a link. The origin of the coordinate system is at the upper left corner of an image, where $+x$ axis points to the upper right corner, and $+y$ axis points to the bottom left corner. Object x value is the central location of a bounding box along the x axis, and object y value is the central location of a bounding box along the y axis. All coordinate values are scaled between 0 and 1, i.e., x value is divided by the width of the image, whereas y value is divided by the height of the training image.

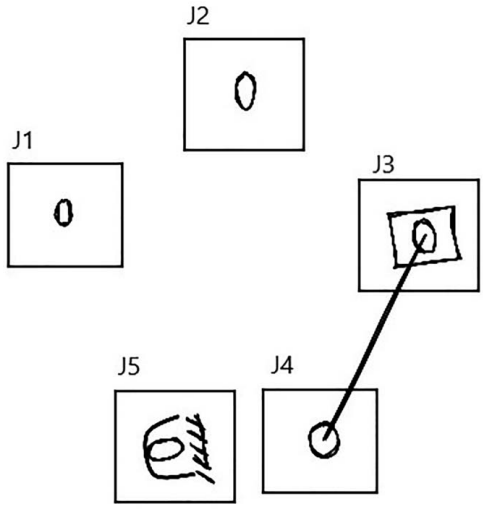

Fig. 3 A training image example with five joints in total, where one is a fixed revolute joint, three are free revolute joints, and one is a prismatic joint. There is a total of one link connecting joints J3 and J4. Actual training images do not have the bounding boxes as well as text written on them. They are shown in this example for a better understanding of the corresponding adjacency matrix.

3 Real-Time Object Detection

In recent years, "You Only Look Once" (YOLO) family [29–32] of convolutional neural networks has demonstrated excellent results with fast object detection. This model uses a single CNN in order to predict both bounding boxes and their class probabilities. This helps speed up the whole detection process and yields very high detection accuracy. In this work, YOLOv4 [32] architecture is used, which was pre-trained on the ImageNet dataset [33] consisting of 1.2 million images of 1000 classes of common objects, such as dogs, cats, etc. Since the images used in ImageNet dataset are markedly different from geometric line diagrams of mechanisms, one cannot simply use the pre-trained network to do predictions. We employed the principle of fine-tuning to retrain YOLOv4 with our synthetic dataset of mechanisms. Thereafter, the fine-tuned model is used for detecting and locating different parts of planar linkage sketches, such as (1) free joints, (2) fixed joints, (3) links, and (4) sliding joints. Processing these results and calculating intersection over union (IoU) values between the links' and joints' bounding boxes, the exact connections between the joints are detected and a full topological model of sketched mechanism is built. We note that an advantage of this approach is that as the better quality CNN models for object detection become available in the future, they could be easily swapped with the one used in this work to get even more accurate results.

3.1 Making Sense of Object Detection. In the course of labeling the dataset, all bounding boxes for the joints were annotated with precision, ensuring that the center of the joint's ellipse is aligned perfectly with the center of the bounding box. This deliberate alignment was undertaken due to the pivotal role the ellipse's center plays in our topology determination algorithm. By precisely ascertaining the positions of the ellipses, we enable the computation of the lengths of links in pixels, thereby ensuring that the interactive bar linkage closely resembles the input sketch.

The object detector is primarily designed to identify individual objects, without inherently recognizing their patterns of interconnection. However, by leveraging the bounding boxes detected by the object detector, it becomes feasible to calculate the precise connections between joints and subsequently determine the mechanism's topology.

Our topology determination algorithm, which is elucidated later in this paper, utilizes the IoU metric [34] for quantifying the extent of overlap between two bounding boxes, thereby defining exact connections. The IoU measurement serves as a valuable metric for assessing the degree of overlap between the detected bounding boxes and their corresponding ground truth counterparts, and it can be calculated as follows:

$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

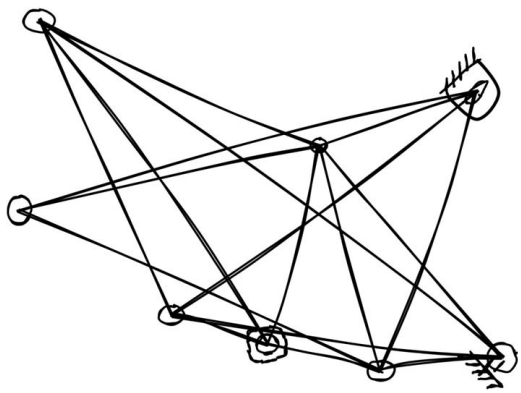

Fig. 4 An unrealistic image with eight joints and 14 links. It is hard to visually differentiate between different links and would result in poor training where it would detect links erroneously.

where the numerator represents the area of overlap, while the denominator signifies the area of union between bounding boxes of shapes A and B. The IoU value is a measure of overlap, with a higher IoU indicating a greater degree of overlap.

The object detector outputs types of joints and their locations indicated by bounding boxes, as well as bounding boxes of the links. Detected bounding boxes are rather big, and this fact can pose a problem; thus, we would like to process initially obtained bounding boxes.

Consider Fig. 5, which illustrates a four-bar linkage alongside its associated bounding boxes. Each bounding box is annotated with two numbers that convey important information. The first number designates the detected class of the mechanism, where 0 signifies a fixed joint, 1 denotes a free joint, 2 represents a link, and 3 indicates a slider joint (though not depicted in this particular example). The second number quantifies the confidence of the model in its prediction; a value closer to 1 indicates a higher degree of confidence.

Black square-shaped bounding boxes correspond to fixed joints, while gray square-shaped boxes represent free joints. Square-shaped bounding boxes with rectangles within represent slider joints, and rectangular bounding boxes indicate the detection of a link within the bounding box.

The challenge associated with utilizing bounding boxes output by the detector becomes evident when examining the bounding box encompassing the longest link in Fig. 5. Calculating the IoU values between this bounding box and those encompassing the joints yields values exceeding 0, erroneously suggesting that this link is connected to all the joints depicted in the example. This clearly illustrates a limitation in using bounding boxes for understanding the mechanism's true topology.

To circumvent this issue, we employ a pre-processing step on the bounding boxes generated by the detector. The pre-processing algorithm operates by taking the bounding boxes of all joints, as produced by the detector, and padding them by 15 pixels. Subsequently, it compiles and stores all pertinent information about these joints in an array. This array includes the x and y coordinates of both the top-left and bottom-right corners of each joint's bounding box, along with its type and corresponding number.

While this approach effectively handles joints, it is not suitable for processing bounding boxes associated with links. Utilizing the same technique for link bounding boxes would mislead the program into assuming that a link is not connected to any other component. Consequently, a distinct technique is employed when dealing with bounding boxes related to links.

The object detector generates bounding boxes for links in such a way that the ends of the links coincide with two diagonal corners of bounding boxes, as shown in Fig. 5. Thus, once the detector produces results for links' bounding boxes, the program detects a line in that bounding box, i.e., the link itself, and calculates the angle of that link with respect to a horizontal line.

All bounding boxes are aligned with the Cartesian x and y axes; thus, there are two possible positions of a link in a bounding box, i.e., a link can either go (1) from the upper left to the lower right corner of a bounding box or go (2) from the lower left to the upper right corner of a bounding box. Calculating the angle of a link in a bounding box, the program assumes that if the angle is more or equal to 0 and less than or equal to 90 deg, then a link goes from the left bottom to the right upper corner of a bounding box, whereas if the angle is more than 90 and less or equal to 180 deg, then a link goes from the left upper to the right lower corner of a bounding box. Once the exact orientation of a link is known, the program fills a separate array with small bounding boxes around link's ends. These bounding boxes are 20×20 pixels wide. In other words, each entry of the array has information about two bounding boxes of each link's ends. This end result of post-processing the detector results is shown in Fig. 6. This process is summarized in Algorithm 1.

Algorithm 1: YOLOv4 results processing

Input: Bounding boxes of all objects in a sketch—$[box_1, box_2, \ldots, box_N]$
Output: Two arrays containing information about bounding boxes of all joints' and links' present in a sketch—**joints**, **links**

Initialize empty arrays of **joints**, **links**
for $i \leftarrow 1$ ***to*** N **do**
  $(x_{tl}, y_{tl}, x_{br}, y_{br}, type, name) = box_i$
  if *type* ***is*** *link* **then**
    link_angle = detect_link_angle(box_i)
    if $0 \leq link\_angle \leq 90$ **then**
      links.add($[x_{tl} - 10, y_{br} - 10, x_{tl} + 10, y_{br} + 10]$, $[x_{br} - 10, y_{tl} - 10, x_{br} + 10, y_{tl} + 10]$)
    else if $90 <$ link_angle ≤ 180 **then**
      links.add($[x_{tl} - 10, y_{tl} - 10, x_{tl} + 10, y_{tl} + 10]$, $[x_{br} - 10, y_{br} - 10, x_{br} + 10, y_{br} + 10]$)
  else
    joints.add($[[x_{tl} + 15, y_{tl} + 15], [x_{br} - 15, y_{br} - 15], type, name]$)
end
return *Arrays with processed bounding boxes' information—* ***joints***, ***links***

4 Topology and Geometry Determination

In this section, we present an algorithm for determining joint–link connections and their geometry. The algorithm begins by iterating through the links. During each iteration, it initializes a zero connections array with a length equal to the number of joints in the sketch

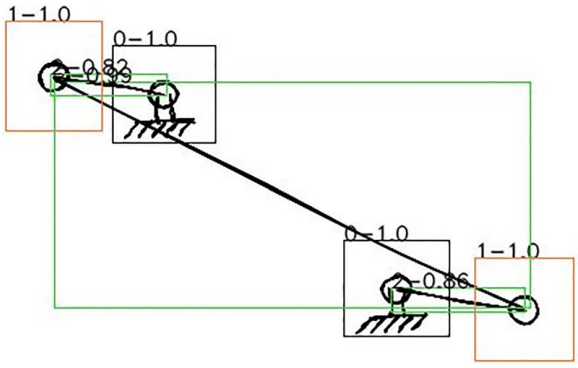

Fig. 5 Unfiltered result produced by the object detector on a sketch generated by our script. Black bounding boxes suggest that it is a fixed revolute joint, gray square box—free revolute joint, and gray rectangular box—link.

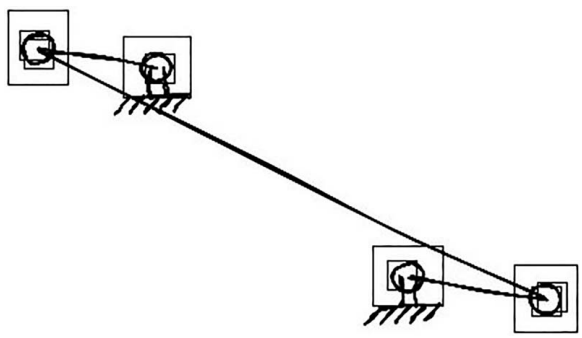

Fig. 6 Same example as shown in Fig. 5, but with processed bounding boxes. Bigger bounding boxes are output by the detector, whereas small bounding boxes relate to links' ends.

and initiates a nested iteration. The nested iteration traverses through the joints array, calculating IoU values between both ends of the link and the current joint. If either of the two calculated IoU values exceeds 0, it indicates a connection between the link and that joint, prompting a change in the corresponding entry in the connections array to 1. After iterating through all the joints, the connections array contains exactly two entries set to 1, as each link connects only two joints, while all other entries remain at 0. By identifying the indices of nonzero entries in the connections array, we can determine which two joints are connected to each other. By looping through all the links, one can identify all the connections present. Algorithm 2 provides a summarized representation of the entire procedure described. The final result of the example depicted in Fig. 5 is illustrated in Fig. 7, where each bounding box indicates the joint type and the order of connection.

Once the topology is defined, it becomes feasible to provide information regarding the lengths of the links, their types, and the joints they connect. Furthermore, comprehensive information about all detected joints, encompassing their type, as well as their x and y coordinates relative to the top-left corner of a canvas, can also be obtained. This information is presented in two distinct tables: one pertaining to links and another pertaining to joints.

The program also generates a square-shaped adjacency matrix [35]. Nonzero entries within this matrix are assigned a value of 1 if two links are connected by a revolute joint and 2 if connected by a prismatic joint. In all other cases, the entries are set to 0, indicating that the respective links are not connected to each other.

Table 1 provides a summary of the total number of links detected, the joints they connect, their respective lengths, and the joint types, whether prismatic or revolute. Meanwhile, Table 2 presents information about the detected joints, encompassing their type and their x and y coordinates relative to the top-left corner of the image. Lastly, Table 3 displays the calculated adjacency matrix.

Algorithm 2: n-bar linkage topology determination

Input: Two arrays with processed bounding boxes: *joints* and *links*
Output: The filtered array of all joint connections present in a sketch—*final_connections*

```
for link in links do
    connections = zero_array
    for index, joint in joints do
        link_one_end = link[0] link_second_end = link[1]
        IoU_1 = calculate_IoU(joint, link_one_end)
        IoU_2 = calculate_IoU(joint, link_second_end)
        if IoU_1 > 0 or IoU_2 > 0 then
            connections[ind] = 1
    end
    non_zero_indexes = find_non_zero_entries(connections)
    final_connections.add(joints[non_zero_indexes])
end
return final_connections
```

While the topology determination algorithm for mechanisms with revolute joints operates as expected, prismatic joints introduce an additional challenge. For instance, the detector results depicted in Fig. 1 suggest the presence of all the joints, but it does not furnish any details regarding the specific connections between the detected slider and the links, i.e., which link the slider is sliding on. To address this challenge, an additional step is performed when a sketch features sliders. Using IoU calculations for the slider, we find that the slider is associated with either of the two links; henceforth called candidate links. Using OpenCV library [36], we can determine the boundaries of the slider as four connected lines of an oriented rectangle. If the orientation of a candidate link matches with the orientation of the longer side of this rectangle, we conclude that link to be the one on which the slider is moving. This further means that the slider forms a connection with the other candidate link via a revolute joint.

This process begins by computing the orientation of each candidate link relative to a horizontal line. Next, we proceed to detect an oriented rectangle within the bounding box of the slider using functions provided in the OpenCV. This rectangle represents the slider. The OpenCV detects all four boundary lines of this rectangle and by locating common endpoints of these lines, a complete rectangle is determined.

After successfully detecting the rectangle, we compare the orientation of the longer side of the rectangle with that of the candidate links. This angle is regarded as the slider angle. The candidate link whose orientation matches with the slider angle is the link associated with the slider. Our assumption is that sliders are generally drawn as rectangles with longer side oriented same as the link on which it is sliding. Conversely, if none of the links' angle match the slider angle within a tolerance of 5 deg, it suggests that the

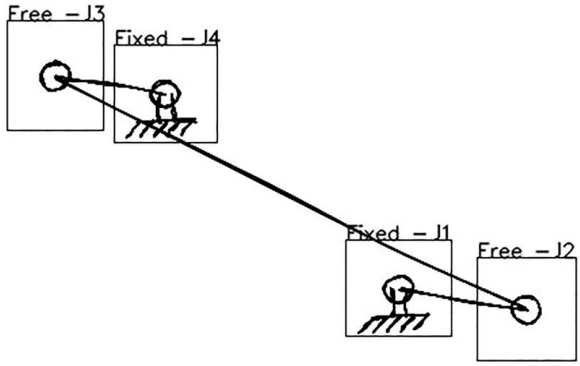

Fig. 7 Final detection result of the given example. Each bounding box gives the information about the joint type and connection order, i.e., J1 is connected to J2, whereas joint J2 is connected to J3, and, likewise, J3 to J4.

Table 1 Detected link information for the example given in Fig. 5

Link #	Link type	Linked joints	Length (pixels)
1	RR binary	J1–J2	95
2	RR binary	J2–J3	389
3	RR binary	J3–J4	83
4	RR binary	J4–J1	224

Table 2 Detected joint information for the example given in Fig. 5

Joint	Description	X	Y
J1	Fixed joint	360	337
J2	Free joint	455	352
J3	Free joint	107	180
J4	Fixed joint	188	193

Table 3 Adjacency matrix for the example given in Fig. 5

	Link 1	Link 2	Link 3	Link 4
Link 1	0	1	1	0
Link 2	1	0	0	1
Link 3	1	0	0	1
Link 4	0	1	1	0

Table 4 Detected link information for the example given in Fig. 1

Link #	Link type	Linked joints	Length (pixels)
1	RP	J2–J4	259
2	Slider	J3–J4	–
3	RR	J1–J3	95
4	Ground link	J1–J2	–

Table 5 Detected joints' information for the example given in Fig. 1

Joint	Description	*X*	*Y*
J1	Revolute fixed	120	353
J2	Revolute fixed	399	359
J3	Revolute free	158	265
J4	Prismatic	158	265

Table 6 Adjacency result for the example given in Fig. 1

	Link 1	Link 2	Link 3	Link 4
Link 1	0	2	0	1
Link 2	2	0	1	0
Link 3	0	1	0	1
Link 4	1	0	1	0

slider is sliding on a fixed link. Due to the diverse nature of hand-drawn styles, the angle between a slider and the link it is sliding on may slightly differ. Hence, it is necessary to account for this variation by allowing both angles a tolerance of 5 deg. This approach enables us to precisely determine the connections between the joints in a mechanism with prismatic joints. Figure 1 visually represents the angles of all the links in red, with the slider's angle depicted in blue. When the slider's angle coincides with the angle of a particular link, it signifies that the slider is sliding on that specific link. Furthermore, Tables 4–6 present comprehensive information about the input sketch mechanism illustrated in Fig. 1.

4.1 Limitations of the Approach. Notably, this approach exhibits scalability for higher *n*-bar linkages consisting solely of revolute joints or revolute-prismatic connections. For four-bar linkages, we capitalize on our kinematic understanding of these mechanisms to interpret the detections produced by our detector. Take, for example, the RRPP mechanism depicted in the bottom-right section of Fig. 9. Our detector identifies two slider joints, a link and a fixed joint. Following the initial detection results and post-processing of bounding boxes, we can establish a connection between the fixed joint and the upper slider joint. Although the curved link connecting the two sliders remains undetected due to the absence of curved links in our training dataset, our kinematic expertise allows us to infer a connection between the two slider joints, as no other type of connection is feasible in this scenario. Consequently, we can apply our algorithm to determine the paths along which these prismatic joints slide by calculating the angle of the longest side within the prismatic joints, thus obtaining the precise topology of this particular mechanism. However, its applicability becomes constrained when dealing with prismatic-prismatic joint connections in an *n*-bar mechanism. This can be overcome by training the neural network with a more exhaustive style of drawing joints and links.

5 Examples and Analysis

This section provides several examples of running the detector on various types of sketches, including script-generated and hand-drawn illustrations.

5.1 Script-Generated Sketches. The object detector was fine-tuned using the dataset of planar *n*-bar linkages generated by our script. The fine-tuning process was carried out on a Dell XPS 15 7590 running on Windows 10 with 32.0 GB of RAM, powered by an Intel Core i7-9750H CPU, and equipped with an NVIDIA GeForce GTX 1650 with 4GB of dedicated memory. The hyperparameters used for fine-tuning the detector included 4000 epochs, which took approximately 28 h to complete, an image size of 416 × 416 pixels, a batch size of 64 sketches, and a learning rate of 0.001. The decision to keep the batch size low was driven by computational constraints. The choice of 4000 epochs was based on experimental observations, as monitoring the training loss, each epoch revealed that the loss stabilized after epoch number 4000. Consequently, it was deemed appropriate to conclude the training process at that point. Furthermore, a low learning rate was employed to mitigate the risk of significantly altering the existing weights during training.

Running the fine-tuned detector on the training dataset took approximately 17 min, while executing it on the testing dataset required about 2 min. Both runs demonstrated excellent results in terms of detecting all links and determining their types and locations. The fine-tuned weights were saved at intervals of 1000 epochs, resulting in four different sets of weights for evaluation—after 1000, 2000, 3000, and 4000 epochs. The highest average mean average precision (mAP) [37], with a threshold value of 0.5, reached 98.76% and 98.70% for the training and testing sets, respectively. This peak performance was attained after 3000 epochs. The outcomes of the model trained for 3000 epochs are presented in Tables 7 and 8. Each table displays the average precision (AP) for every class, along with counts of true positives (TPs) and false positives (FPs). The results revealed that at 4000 epochs, the model's accuracy diminished for both datasets, affirming the appropriateness of our decision to halt the training process to prevent over-fitting. Comparing these results to the ones presented in Ref. [20], we observe a nearly 40% improvement in the average precision results. Additionally, while their approach is scalable to higher-order mechanisms, it remains unclear how prismatic joints are treated or even detected when no revolute joints are drawn on them. Moreover, the post-processing algorithm presented in the paper that corrects the initial results of the mechanism detection might pose challenges as the number of joints increases and could also require considerably more time [20].

These outcomes indicate that the model achieved nearly flawless detection of joints; nevertheless, detecting links presented a more challenging task, as the average precision did not approach 100%, and a significant number of links were erroneously detected. The high mAP results for both datasets imply that the model effectively

Table 7 Results of running the detector on the training set after 3000 epochs

Class	AP	TP	FP
Fixed joint	100.00%	38,976	364
Free joint	100.00%	93,691	1773
Link	93.47%	105,133	16,094
Slider joint	100.00%	19,516	53

Table 8 Results of running the detector fine-tuned for 3000 epochs on the test set

Class	AP	TP	FP
Fixed joint	100.00%	3929	43
Free joint	100.00%	9329	173
Link	93.17%	10,577	1700
Slider joint	100.00%	1968	7

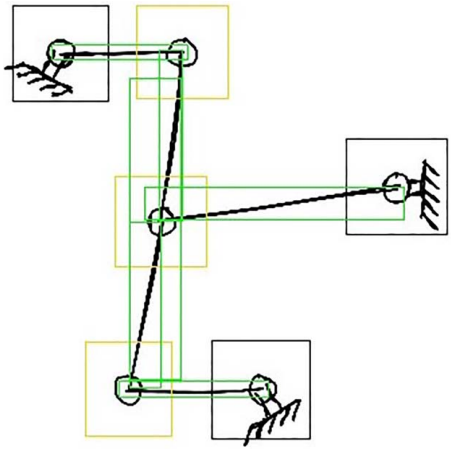

Fig. 8 An example result of running the detector on a sketch generated through our script similar to the training dataset sketches

learned to identify various components accurately, as well as to generate bounding boxes resembling the ground truth boxes. Figure 8 illustrates the outcome of applying the detector to a sketch generated using our script. Importantly, this sketch was not part of either the training or testing datasets.

Running the detector on testing sets with varying stroke and background colors and different stroke thicknesses revealed that alterations in stroke color, background color, and stroke thickness levels 1, 2, and 3 had a minimal impact on accuracy results. However, increasing the stroke thickness from level 4 to 6 led to a decrease in accuracy precision. The average mAP values for each testing dataset are presented in Table 9. One approach to improving accuracy with these thickness levels would involve retraining the model using these images. However, given the low likelihood of real sketches having such high stroke thickness levels, we opted not to fine-tune our model accordingly.

Table 9 Results of running the fine-tuned detector on different testing datasets

Testing set	mAP
Colored stroke on white background	97.75%
Colored background with black stroke	97.58%
Mix of colored background and stroke	96.69%
Stroke level 1	98.29%
Stroke level 2	98.70%
Stroke level 3	98.46%
Stroke level 4	94.80%
Stroke level 5	74.55%
Stroke level 6	64.87%

5.2 True Hand-Drawn Sketches. Training and test images generated by the Scribble library simulate hand-drawn sketches, but they may not perfectly replicate the handwriting styles of all individuals. Therefore, it was imperative to assess the detector's performance on genuine hand-drawn sketches. To facilitate this, we developed a straightforward graphical user interface program using the PYTHON library tkinter. This program allowed users to create and save various black and white sketches, with a focus on bar sketches for our purposes.

We employed this interface to generate and evaluate different types of planar *n*-bar linkage sketches. Six of the most commonly used four-bar linkages were examined using the detector, and the results are displayed in Fig. 9. Additionally, Fig. 10 features two random *n*-bar linkages. It is worth noting that the examples were drawn without strict adherence to whether the resulting mechanism would be physically valid or not. Our primary objective was to assess the model's ability to accurately detect objects in hand-drawn sketches.

The testing dataset used to evaluate the detector's accuracy on actual hand-drawn sketches was relatively small, consisting only 30 images. This limited size was due to the challenge of gathering a larger number of hand-drawn samples compared to the training data. Upon visual inspection of the results, it became evident that

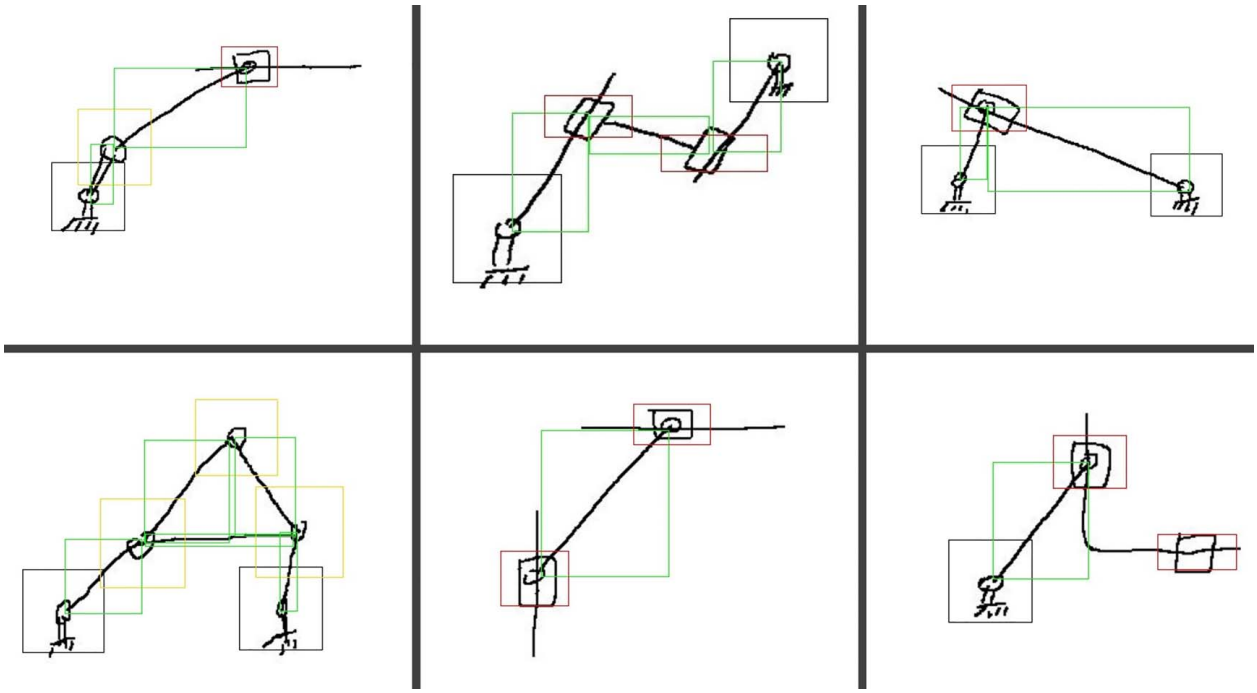

Fig. 9 Example of running the detector on hand-drawn sketches of commonly used four-bar linkages: top-left—RRRP, top middle—RPPR, top right—RRPR; bottom left—RRRR with a coupler link shown as three different binary links, bottom middle—PRRP, bottom-right—RRPP. All objects were detected correctly.

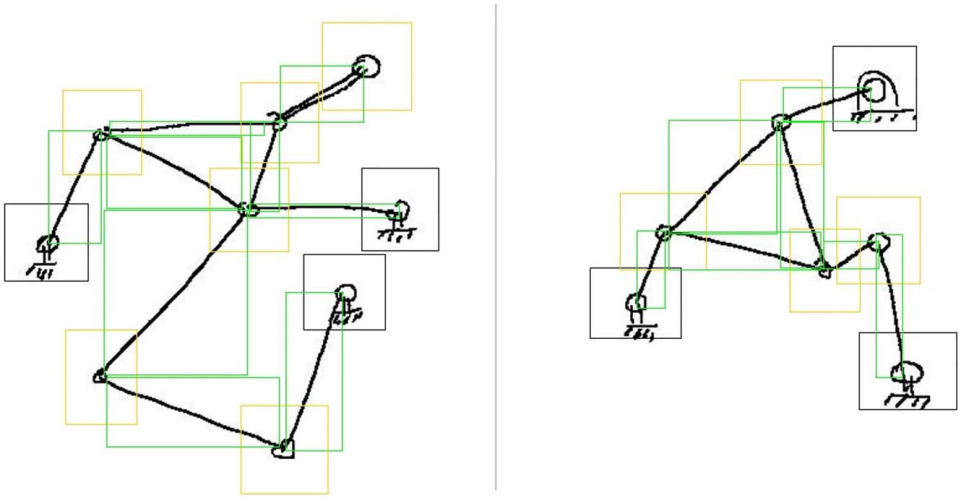

Fig. 10 Example of running the detector on two hand-drawn sketches. All objects were detected correctly.

Table 10 Results of running the detector fine-tuned for 3000 epochs on the test set of hand-drawn sketches

Class	Total #	TP	FP
Fixed joint	60	60	0
Free joint	119	119	0
Link	249	230	5
Slider joint	35	35	0

the fine-tuned model performs admirably in detecting objects and their types within hand-drawn sketches depicting n-bar linkages.

It is noteworthy that during training, the detector was taught to recognize a single straight line between two joints as a link. However, some individuals sketch a link as two separate lines between joints. For instance, the left sketch in Fig. 10 features such a link in the top right corner, and the detector correctly identified it. This suggests that our model can effectively handle cases that it has not encountered during training.

Table 10 provides an overview of the dataset, showcasing the total number of joints and links within it and how many were accurately detected. All joints were correctly identified, although a few links were not detected, and there were five false positives.

6 Conclusions

This study introduced an innovative framework that combines real-time object detection with rigid body kinematics to facilitate the digitization and organization of hand-drawn linkage mechanism sketches. Specifically, we proposed a methodology for generating synthetic images of such sketches and demonstrated how the results obtained from a convolutional neural network (CNN)-based object detector can be integrated with the topological understanding of mechanisms to achieve the desired outcomes. Our experimental results show the effectiveness of this approach, and it holds potential for further enhancements in machine learning research, while still leveraging the extensive knowledge available in the field of kinematics. Looking ahead, future research directions may involve extending this approach to non-planar sketches and textbook-style images of linkage sketches.

Acknowledgment

This work has been financially supported by The National Science Foundation under research grants #CMMI-1563413 and STTR phase II #2126882 to co-author Purwar who also holds stocks in Mechanismic Inc. The research findings included in this publication may or may not necessarily relate to the interests of Mechanismic Inc. The terms of this arrangement have been reviewed and approved by Stony Brook University in accordance with its policy on objectivity in research.

All findings and results presented in this paper are those of the authors and do not represent those of the funding agencies.

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The data and information that support the findings of this article are freely available.[3]

References

[1] Huang, Z., 2020, "Deep-Learning-Based Machine Understanding of Sketches: Recognizing and Generating Sketches With Deep Neural Networks," Master's thesis, University of California at Berkeley, Berkeley, CA.
[2] Güçlütürk, Y., Güçlü, U., van Lier, R., and van Gerven, M. A. J., 2016, "Convolutional Sketch Inversion," European Conference on Computer Vision, Amsterdam, The Netherlands, Oct. 11–14.
[3] Ellis, K., Ritchie, D., Solar-Lezama, A., and Tenenbaum, J. B., 2017, "Learning to Infer Graphics Programs From Hand-Drawn Images," CoRR, abs/1707.09627.
[4] Simo-Serra, E., Iizuka, S., Sasaki, K., and Ishikawa, H., 2016, "Learning to Simplify," Trans. Graph., **35**(4), pp. 1–11.
[5] Oh, S., Jung, Y., Kim, S., Lee, I., and Kang, N., 2019, "Deep Generative Design: Integration of Topology Optimization and Generative Models," ASME J. Mech. Des., **141**(11), p. 111405.
[6] Pu, J., Lou, K., and Ramani, K., 2005, "A 2d Sketch-Based User Interface for 3d CAD Model Retrieval," Comput. Aided Des. Appl., **2**(6), pp. 717–725.
[7] Willis, K. D. D., Jayaraman, P. K., Lambourne, J. G., Chu, H., and Pu, Y., 2021, "Engineering Sketch Generation for Computer-Aided Design," CoRR, abs/2104.09621.
[8] Para, W. R., Bhat, S. F., Guerrero, P., Kelly, T., Mitra, N. J., Guibas, L. J., and Wonka, P., 2021, "Sketchgen: Generating Constrained CAD Sketches," CoRR, abs/2106.02711.
[9] Kazi, R. H., Grossman, T., Cheong, H., Hashemi, A., and Fitzmaurice, G., 2017, "DreamSketch," The 30th Annual ACM Symposium on User Interface Software and Technology, Québec City QC Canada, Oct. 22–25.
[10] Murugappan, S., Vinayak, Ramani, K., and Yang, M. C., 2011, "APIX: Analysis From Pixellated Inputs in Early Design Using a Pen-Based Interface." ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Washington, DC, Aug. 28–31.

[3]See Note 2.

[11] Murugappan, S., and Ramani, K., 2009, "FEAsy: A Sketch-Based Interface Integrating Structural Analysis in Early Design," 29th Computers and Information in Engineering Conference, San Diego, CA, Aug. 30–Sept. 2.
[12] Nie, Z., Jiang, H., and Kara, L. B., 2020, "Stress Field Prediction in Cantilevered Structures Using Convolutional Neural Networks," ASME J. Comput. Inf. Sci. Eng., **20**(1), p. 011002.
[13] Nobari, A. H., Srivastava, A., Gutfreund, D., and Ahmed, F., 2022, "LINKS: A Dataset of a Hundred Million Planar Linkage Mechanisms for Data-Driven Kinematic Design," International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, St. Louis, MO, Aug. 14–17.
[14] Deshpande, S., and Purwar, A., 2019, "A Machine Learning Approach to Kinematic Synthesis of Defect-Free Planar Four-Bar Linkages," ASME J. Comput. Inf. Sci. Eng., **19**(2), p. 021004.
[15] Deshpande, S., and Purwar, A., 2020, "An Image-Based Approach to Variational Path Synthesis of Linkages," ASME J. Comput. Inf. Sci. Eng., **21**(2), p. 021005.
[16] Deshpande, S., and Purwar, A., 2019, "Computational Creativity Via Assisted Variational Synthesis of Mechanisms Using Deep Generative Models," ASME J. Mech. Des., **141**(12), p. 121402.
[17] Purwar, A., and Chakraborty, N., 2023, "Deep Learning-Driven Design of Robot Mechanisms," ASME J. Comput. Inf. Sci. Eng., **23**(6), p. 060811.
[18] Eicholtz, M., Kara, L. B., and Lohn, J., 2014, "Recognizing Planar Kinematic Mechanisms From a Single Image Using Evolutionary Computation," Genetic and Evolutionary Computation Conference, Vancouver, BC, Canada, July 12–16, ACM.
[19] Eicholtz, M., and Kara, L. B., 2015, "Intermodal Image-Based Recognition of Planar Kinematic Mechanisms," J. Vis. Lang. Comput., **27**(1), pp. 38–48.
[20] Eicholtz, M., and Kara, L. B., 2015, "Characterizing the Performance of an Image-Based Recognizer for Planar Mechanical Linkages in Textbook Graphics and Hand-Drawn Sketches," Comput. Graph., **52**(C), pp. 1–17.
[21] Papageorgiou, C., Oren, M., and Poggio, T., 1998, "A General Framework for Object Detection," Sixth International Conference on Computer Vision, Bombay, India, Jan. 7, pp. 555–562.
[22] Dalal, N., and Triggs, B., 2005, "Histograms of Oriented Gradients for Human Detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Diego, CA, June 20–26.
[23] Evgeniou, T., and Pontil, M., 2001, "Support Vector Machines: Theory and Applications," *Machine Learning and Its Applications*, G. Paliouras, V. Karkaletsis, and C. D. Spyropoulos, eds., Springer, Berlin, pp. 249–257.
[24] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T., 2002, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," IEEE Trans. Evol. Comput., **6**(2), pp. 182–197.
[25] Purwar, A., Deshpande, S., and Ge, Q. J., 2017, "MotionGen: Interactive Design and Editing of Planar Four-Bar Motions Via a Unified Framework for Generating Pose- and Geometric-Constraints," ASME J. Mech. Rob., **9**(2), p. 024504.
[26] Mechanismic Inc., 2022, "MotionGen Pro", St. James, NY, http://www.motiongen.io
[27] Wullschleger, J., and Vucetic, O., 2017, p5.scribble.js, https://github.com/generative-light/p5.scribble.js
[28] Wood, J., Isenberg, P., Isenberg, T., Dykes, J., Boukhelifa, N., and Slingsby, A., 2012, "Sketchy Rendering for Information Visualization," IEEE Trans. Vis. Comput. Graph, **18**(12), pp. 2749–2758.
[29] Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A., 2015, "You Only Look Once: Unified, Real-Time Object Detection," CoRR, abs/1506.02640.
[30] Redmon, J., and Farhadi, A., 2016, "YOLO9000: Better, Faster, Stronger," CoRR, abs/1612.08242.
[31] Redmon, J., and Farhadi, A., 2018, "YOLOv3: An Incremental Improvement," CoRR, abs/1804.02767.
[32] Bochkovskiy, A., Wang, C., and Liao, H. M., 2020, "Yolov4: Optimal Speed and Accuracy of Object Detection," CoRR, abs/2004.10934.
[33] Krizhevsky, A., Sutskever, I., and Hinton, G. E., 2012, "Imagenet Classification With Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems*, F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger, eds., Curran Associates, Inc., Red Hook, NY, pp. 1097–1105.
[34] Rezatofighi, S. H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I. D., and Savarese, S., 2019, "Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression," CoRR, abs/1902.09630.
[35] Tsai, L.-W., 2001, *Mechanism Design Enumeration of Kinematic Structures According to Function*, CRC Press, Boca Raton, FL.
[36] Culjak, I., Abram, D., Pribanic, T., Dzapo, H., and Cifrek, M., 2012, "A Brief Introduction to Opencv," 2012 Proceedings of the 35th International Convention MIPRO, Opatija, Croatia, May 21–25, pp. 1725–1730.
[37] Henderson, P., and Ferrari, V., 2016, "End-to-End Training of Object Class Detectors for Mean Average Precision," CoRR, abs/1607.03476.