State-Aware Configuration Detection for Augmented Reality Step-by-Step Tutorials

Ana Stanescu [1*] Peter Mohr [1] Mateusz Kozinski [1] Shohei Mori [1] Dieter Schmalstieg [1,3] Denis Kalkofen [2,1†]
[1] Graz University of Technology [2] Flinders University [3] VRVis

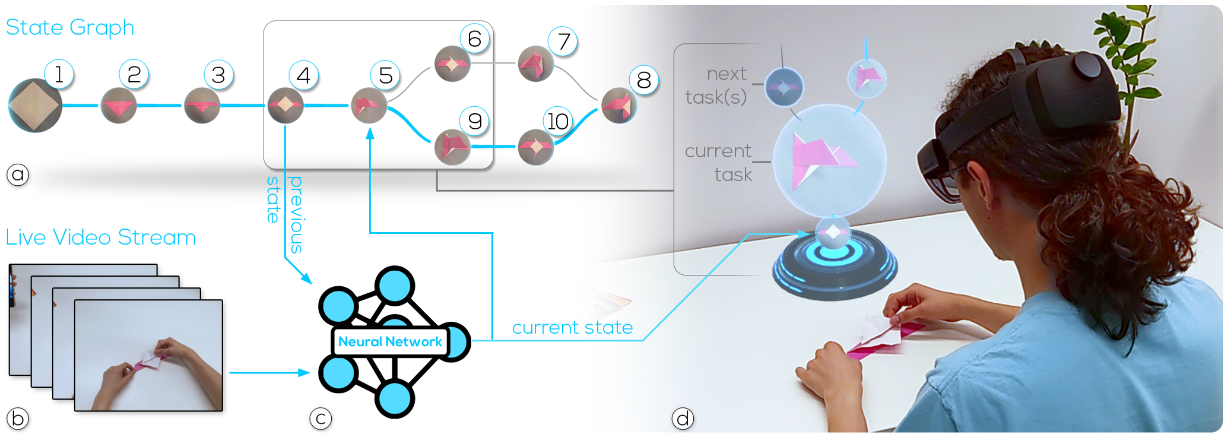

Figure 1: Image-based configuration detection enables automatically progressing augmented reality tutorials based on user performance. To robustly distinguish between object states, we introduce (c) a state-aware neural network, which is trained using (a) a state graph in addition to images that show the object configurations. State-aware configuration detection improves existing approaches by incorporating the possible configuration order, which is given by the structure of the state graph. (b) We integrate our method into an augmented reality system to detect the object configuration of step-by-step tutorials in real time. (d) This enables presenting an interactive visualization of the state graph, which automatically selects the current task (shown enlarged).

Abstract

Presenting tutorials in augmented reality is a compelling application area, but previous attempts have been limited to objects with only a small numbers of parts. Scaling augmented reality tutorials to complex assemblies of a large number of parts is difficult, because it requires automatically discriminating many similar-looking object configurations, which poses a challenge for current object detection techniques. In this paper, we seek to lift this limitation. Our approach is inspired by the observation that, even though the number of assembly steps may be large, their order is typically highly restricted: Some actions can only be performed after others. To leverage this observation, we enhance a state-of-the-art object detector to predict the current assembly state by conditioning on the previous one, and to learn the constraints on consecutive states. This learned 'consecutive state prior' helps the detector disambiguate configurations that are otherwise too similar in terms of visual appearance to be reliably discriminated. Via the state prior, the detector is also able to improve the estimated probabilities that a state detection is correct. We experimentally demonstrate that our technique enhances the detection accuracy for assembly sequences with a large number of steps and on a variety of use cases, including furniture, Lego and origami. Additionally, we demonstrate the use of our algorithm in an interactive augmented reality application.

Index Terms: Human-centered computing—Human computer interaction (HCI)—Interaction paradigms—Mixed / augmented reality; Computing methodologies—Machine learning—Learning settings—Learning from demonstrations

*e-mail: ana.stanescu@icg.tugraz.at
†e-mail: kalkofen@icg.tugraz.at

1 Introduction

Showing visual assembly instructions directly in place is a powerful use case of Augmented Reality (AR). It overcomes the need for mentally complex hand-eye coordination when following instructions presented on traditional 2D screens [4], as real and virtual objects can co-exist within the AR space. The ability to show visual instructions registered in 3D to the physical object can significantly reduce the cognitive load required to follow instructions [8, 12]. Early systems required the user to manually advance to the next step in the tutorial [42]. However, the need to confirm the completion of every step is generally perceived as inconvenient or even distracting. Users prefer that any object manipulation, e.g., assembly, disassembly, or repositioning of parts, automatically triggers feedback, either by advancing to the next instruction, or by pointing out wrong actions.

Such automatic instruction triggering requires the capacity to detect the current assembly state. Contemporary approaches to state detection rely on analysis of depth and shape [11, 32], template matching [34], or visual 6DOF object tracking [40]. However, even state-of-the-art object detectors struggle when trying to distinguish similar configurations, which only differ in a single part. The problem is exacerbated when the number of states that must be distinguished grows. Consequently, previous real-time state detection methods were limited to rather simple assemblies with only a few large parts and easily observable differences between states.

Our goal is to lift this limitation and make state detection scale to objects composed of a large number of small parts and complex assembly procedures with many steps. To achieve this goal, we take advantage of the fact that the space of assembly states is highly structured. We formalize our idea by representing the state space as a directed acyclic graph, with nodes corresponding to states and edges corresponding to actions (Figure 1). For disassembly, the direction of the edges is simply inverted. An assembly graph encodes constraints on exploring the state space: If a pair of states is not connected with an edge in the graph, the second state cannot be attained from the first one with a single action.

Consequently, being able to distinguish between the complete set of states is typically not necessary – if the previous assembly state is known, and state recognition is fast enough to observe the user's actions in real time, only the previous state and its direct descendants in the assembly graph must be considered. In Figure 1, given the previous state 4, visual recognition can be limited to distinguishing between that state and its neighbors, giving a three-element set of candidate states $\{3,4,5\}$, as opposed to the ten-element set of all assembly states. In non-trivial assemblies, the total number of states can rise exponentially with the number of parts, while only a small number of different parts can be installed at any given state.

In practice, instead of imposing predefined constraints on possible pairs of consecutive states, we propose to learn the relations between consecutive states from data. We explore this idea by conditioning a state-of-the-art object detector, YOLOv7 [35] on the previously observed state. We integrate the learned representation of the previous state into the architecture of the detector by enhancing it with the context of the current image and adding it to the feature map just before the classifier head. To minimize overfitting, we apply dropout [31] to the previous state representation when training the conditional detector.

While we motivate our conditional detector with the need for detecting assembly states, it is not limited to assembly procedures or rigid objects. We show that our work can be deployed in any AR tutorial that relies on a procedure involving recognizable physical states. We evaluate our work on the state-labeled dataset used in the approach by Liu et al. [14], and contribute three new data sets, Lego, Origami, and Engine Block. Our experiments demonstrate that, when the number of states becomes large, our detector consistently outperforms plain visual recognition. At the same time, it is as easy to train as the purely visual baseline and attains nearly the same frame rate at test time. This level of performance allowed us to create a HoloLens 2 application demonstrating guidance for folding Origami and completing Lego constructions. In summary, our work makes the following technical contributions:

- A new approach to assembly state detection by conditioning on the previous state,
- An evaluation of several methods to incorporate the previous state representation in the detector architecture,
- Three data sets of complex assembly sequences, and
- A practical demonstration of our state detector in a real-time AR application.

To our knowledge, we propose the first assembly state detection that does not only rely on visual recognition, but also implicitly learns information to use the state space structure in a neural network.

2 Related work

Our work contributes to the state of the art on interactive AR tutorials. By identifying the current state in a procedural tutorial, an AR visualization can be automatically adapted. Such a detection of the current state is a variant of object detection, i.e., detecting a state corresponds to detecting an object as a particular configuration of parts. Therefore, we review previous work on authoring AR step-by-step tutorials with a focus on state detection and image-based object detection for real-time applications.

2.1 Step-by-step tutorials

Early work on 2D video tutorials demonstrates the importance of temporal segmentation and automatic progression to guide a user through the steps of an image manipulation tutorial [26]. Later, Petersen et al. [24, 25] presented an approach for authoring step-by-step instructions using video segmentation for AR applications. By training classifiers on the segmentation result, the approach is able to follow a user's performance. Damen et al. [7] extend upon video segmentation by providing additional information about the actions required in each step. While approaches to automatic video segmentation have been used to observe a user's progress, it is typically necessary that the user assume a camera pose similar to the one used during training to understand the instructions.

Such viewpoint restrictions imposed by 2D video tutorials can be overcome in AR by presenting the visual information in 3D. Examples include approaches to authoring 3D tutorials from image [21, 23, 36] and video data [22, 40]. However, these approaches process their input data sequentially in an offline manner, which makes them unsuitable for real-time state detection.

With 3D tracking, one can overcome the view restrictions of approaches based on 2D video. Some approaches provide such tracking by instrumenting all parts of an assembly with fiducials [29, 38, 42]. By tracking both the user and the manipulated parts in 3D, a tutorial can be created from a user demonstration [5]. Alas, the requirement of fitting every part with a fiducial is cumbersome and unsuitable in many situations.

2.2 State detection

The key to step-by-step AR tutorials with automated user feedback lies in detecting the current state and suggesting the next known step based on procedures configured in advance (e.g., with an assembly or state graph). The procedures can be defined in a desktop tool [40], in AR space by tagging objects [5, 27], or by demonstrating the procedures [16, 32]. Given such a set of predefined object configurations, the system must know the current configuration of the observed object. Approaches that offer assembly guidance without state detection require the user to manually transition to the next state [18, 40, 44]. We do not further discuss these manual approaches.

An early work by Gupta et al. [11] evaluates the color and structure of Duplo pieces to identify when to advance to the next state automatically. This work was one of the first to use state detection to monitor the user's performance in a tutorial. However, their method is only suited for Duplo bricks, as it assumes that the geometric structure consists of blocks that have the exact dimensions of Duplo bricks. Similarly, Miller et al. [20] guide the assembly of objects by precisely fitting an object into a voxel grid. Wu et al. [39] propose an interactive system that can detect states via RGB-D object tracking. According to a predefined assembly graph and two object-relative poses, the system suggests the next step. Wang et al. [34] estimate the upcoming step based on a posterior probability with a given current state. This formulation does not support multi-part assemblies. Bhattscharya et al. [2] use traditional computer vision algorithms to identify assembly states via point clouds. This approach restricts the partial assembly to be fixed in the environment.

To overcome the issues in classic computer vision, recent approaches rely on neural networks. The method by Liu et al. [14] introduces an attention-based module integrated into Faster-RCNN [30] to distinguish state objects. They show the performance of their network on two objects, a table and a fender object, each having only a few parts. Also, their approach uses synthetic data for training and captured data for testing. Another method by Zhou et al. [43] focuses on significant regions of interest. Their approach is based on extracting multiple regions of interest after a complex preparation procedure involving the pre-training of the interest regions. At runtime, the neural network classifies regions that have been re-identified into object states. The observed viewpoints must be similar to the onesof the recordings. Less than 20 states are handled.

In comparison to all the methods above, we aim to detect complex object configurations (e.g., over a hundred states), as well as to allow for any kind of discrete states, not just rigid objects.

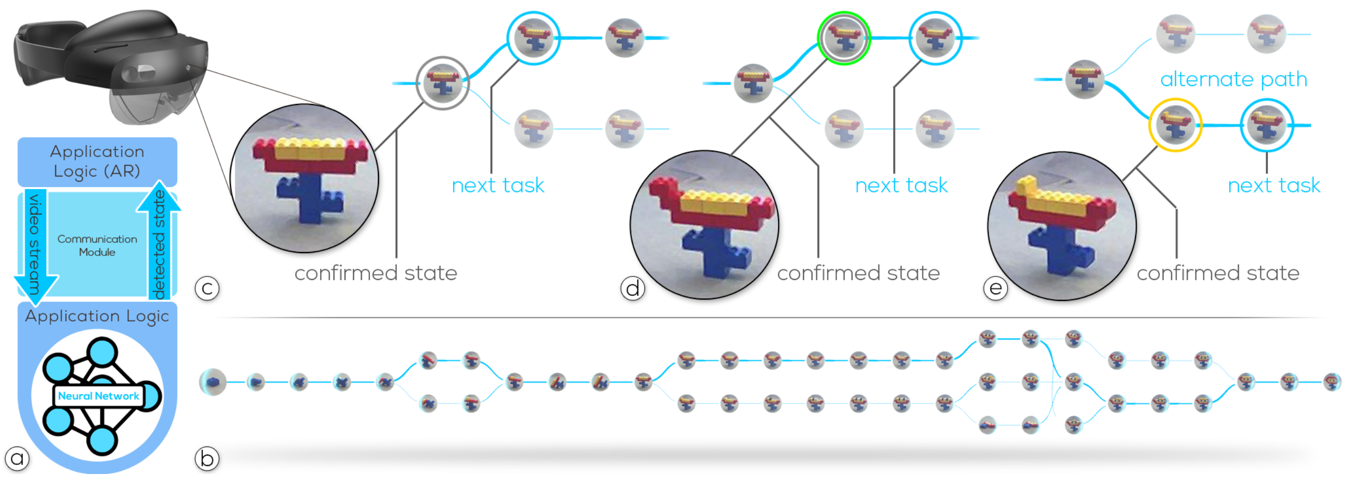

Figure 2: Overview. (a) A user with an AR headset observes a working area containing the object to be manipulated. While the object's configuration changes through various states, a video is captured with a head-worn camera and streamed to a server. The server detects the current state and generates a feedback visualization indicating the actions requires to follow in the current (and if available in the next) step. (b) Our method learns to robustly detect object states from the camera stream by incorporating the state graph of the corresponding step-by-step tutorial. (c) This enables to select the tasks to perform in each step, (d) it allows to confirm the correct result of a given task, and (e) it enables to show when the user chooses to perform an alternative step within the state graph.

2.3 Object detectors

In order to suggest future assembly actions, an AR tutorial system needs to detect the workpiece and recognize its assembly state. We address this problem as an instance of object detection, a classical computer vision task. The task is, therefore, to predict a bounding box and a class label for each object present in a test image. The technical challenge behind object detection stems from the number of objects not known a priori and potentially many overlaps between bounding boxes corresponding to different objects. One way to approach this problem is to separate the generation of hypothetical bounding boxes from the classification. For example, R-CNN [10] and Fast R-CNN [9] use a deep neural network to classify image regions extracted with a selective search algorithm. Faster R-CNN [30] improves these approaches by integrating both tasks in a single deep architecture that extracts candidate bounding boxes and instantly classifies them. An alternative approach is to forgo predicting the bounding-box candidates in favor of a large but fixed set of candidate locations uniformly covering the image. For each candidate, YOLO [28] predicts how likely it is to represent an object, its class, and a displacement between the candidate bounding box and the one tightly circumscribing the object. The correction to the bounding box coordinates is predicted simultaneously with the object class, which contributes to the efficiency of this algorithm. We rely on YOLOv7 [35], an improvement of YOLO that is widely used, fast at test time, and easy to train. As described in Section 3, we enhance it to condition the predicted assembly state on the previous state and to make it learn the constraints on consecutive states.

3 Method

AR tutorials are typically organized in terms of the states of the procedure the user is trying to accomplish. For example, when folding an Origami bird, two adjacent states can be 'wing folded' and 'wing unfolded.' The instructions displayed to the user depend on the current state. In the 'wing unfolded' state, the system would advise the user to fold the wings, whereas in the 'wing folded' state the instruction might ask to 'fold the beak.' User actions, such as folding the wing, cause state transitions. Upon detecting the new state, the AR system should update the instruction. We show an overview of our proposed method in Figure 2.

3.1 State graph

State detection is complicated by the fact that, in real procedures, the number of states may be very large. This observation is especially true for assembly tasks, where actions consist of installing or removing parts from the assembled object. In this case, the total number of states is lower-bounded by 2^k, where k is the size of the largest subset of parts that can be installed independently from one another. We claim that this difficulty can be alleviated if the current state is known, because the number of states that can be accessed from the current state with a single operation is typically small.

To make our argument more formal, we represent the structure of the state space in the form of an assembly graph $G = (V,E)$, where the vertices $v \in V$ are assembly states and an edge $(v,v') \in E$ connects nodes $v,v' \in V$ if there exists an assembly action that takes the procedure from state v to state v'. Since a single action can only advance the procedure from the previous state to one of its neighbors in the graph, the knowledge of the previous state, the predecessor v_0, can greatly facilitate recognition of the current state.

More formally, the set C of candidate states has to be considered as $C(v_0) = \{v_0\} \cup v'$ s.t. $(v,v') \in E\}$. In the case of assembly tasks, for any $v \in V$, the cardinality of $C(v)$ is *upper-bounded* by the number of parts n since each action available at state v either installs a part not yet included in the assembly, or removes a part installed before. The set of candidates is therefore smaller than the set of all states V which grows at least linearly, but typically exponentially, with the number of parts. In real applications, the set of candidates is much smaller than the number of parts, because installing some parts requires others to be in place and not all combinations are physically possible. Therefore, knowledge of the previous state greatly reduces the uncertainty of the current one. Of course, this expectation only holds when no more than one action can be completed between consecutive state detections. Fortunately, this assumption is satisfied in practical AR scenarios.

To make the knowledge of the previous state benefit the detection of the current one, we design an assembly state detector that conditions its predictions on the previously observed state. We describe its architecture, the training procedure, as well as an example AR application in the following sections.

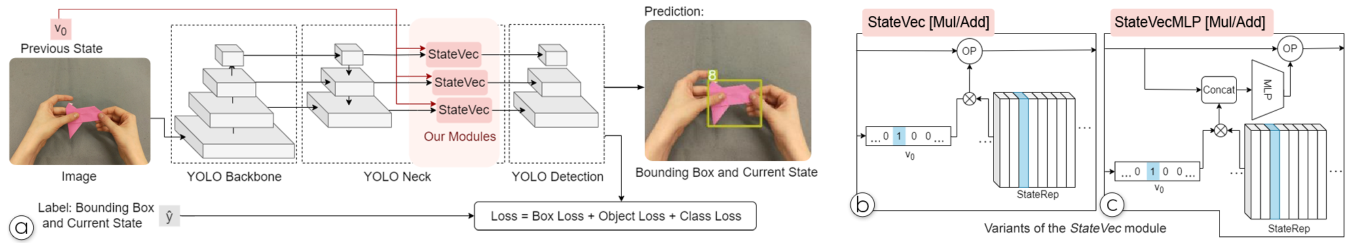

Figure 3: Network architecture. (a) We augment the YOLOv7 detector [35] to condition the assembly state detection on the previous state. The standard YOLOv7 architecture is shown in grey, and our modifications are highlighted in red. We use our *StateVec* module to inject the encoding of the previous state into the feature map between the 'neck' and the 'head' of the architecture. We experiment with two different forms of the *StateVec* module, (b) one that learns the representation of the previous state directly, (c) and one that enhances the learned representation with the context of the image, by concatenating the representation with image features and passing it through a multi-layer perceptron (MLP).

3.2 Detector architecture

As a starting point, we chose YOLOv7 [35], a state-of-the-art deep neural network for object detection from images. Our choice was guided by its high frame rate and availability in the ONNX.AI standard for deployment on mobile devices. The high-level architecture of the entire family of YOLO detectors, represented in grey in Figure 3a, consists of three subnetworks: the backbone, the neck, and the detection head [3]. The backbone extracts features from the image, the neck aggregates features at different resolutions, and the head produces predictions for individual bounding-box candidates. To make the detector conditional on the previous state, we follow the late-fusion approach [33] and inject the representation of the previous state before the head sub-network, as highlighted in red in Figure 3a. The rationale is that, at this stage of the architecture, features represent high-level information about image content, and information about the previous state is at a similar level of abstraction. Each stage of the network processes features at three different resolutions, represented in Figure 3a by the horizontal arrows, and we inject the previous state into each of the resolution levels.

Injecting previous state into the network. We design a deep network layer *StateVec* to introduce the information of the previous state into a feature map ϕ, the output of the 'neck' of the deep network. ϕ is a table of width W and height H, which depend on the size of the input image. Its entries ϕ_{wh} are feature vectors of length n_f, typically 128, 256, or 512. For each (w,h), s.t. $0 < w < W$ and $0 < h < H$, ϕ_{wh} represents a rectangular region of the input image, centered at pixel $w_0 + ws_w, h_0 + hs_h$, where the offsets w_0, h_0 and strides s_w, s_h depend on the details of the architecture. Our goal is to produce previous-state-aware features ϕ'_{wh} by injecting into ϕ_{wh} a learned representation of the previous state, denoted $r(v_0)$. For compatibility with ϕ_{wh}, we give $r(v_0)$ the form of a vector of length n_f. We denote the injection operation as $\phi'_{wh} = g(\phi_{wh}, r(v_0))$. Inspired by previous work on fusing non-visual information into convolutional neural networks, given classes that are visually similar [6,17,33,41], we test two forms of g: the additive $g^+(\phi_{wh}, r) = \phi_{wh} + r$, and the multiplicative $g^*(\phi_{wh}, r) = \phi_{wh} \circ \sigma(r)$, where $\circ$ denotes the element-wise product, and σ is the sigmoid function. g^+ can be interpreted as the simplest way of inserting information into a feature, while g^* can be thought of as attenuating selected feature components.

Representing states. In the basic version of our approach, we simply learn the previous state representation $r(v_0)$ for each $v_0 \in V$. To this end, we parameterize the *StateVec* module with a matrix R of size $|V| \times n_f$ and make each of its rows represent one state. The previous state v_0 is encoded as a one-hot vector, and the state representation is computed as $r(v_0) = v_0^\intercal \mathrm{R}$. This approach is presented in the Figure 3b. In addition to the basic approach, we also experimented with enhancing $r(v_0)$ with the context of the image before injecting it into the feature map. We implemented this enhancement in the form of a shallow network m that concatenates $r(v_0)$ with ϕ_{wh} and propagates the resulting vector through a fully connected network with one hidden layer, yielding a context-enhanced representation of the previous state $r'(v_0) = m(r(v_0), \phi_{wg})$. We illustrate this approach in Figure 3c.

3.3 Training the detector

The standard YOLOv7 is trained on a data set of pairs $(x, \hat{y})$, where x denotes the input image and $\hat{y}$ is the corresponding ground truth. Since we extended the detector to condition on the previous state v_0, we also needed to augment each training sample to contain the previous state, yielding a training set of triplets $(x, v_0, \hat{y})$. In theory, to simulate test-time conditions, v_0 should be computed by running the detector on a sequence of frames preceding x in the training recording. In practice, this approach is very memory-intensive and time-consuming.

In the interest of efficiency, we adopted a simplified training procedure, in which we set v_0 to the one-hot vector representing the correct class of the previous frame. This is shown in Figure3b and c. Since the number of edges in G can be large, and the assembly procedure can be completed without traversing every edge, a large number of recordings may be needed to observe each valid combination of the previous and current state for a sufficient number of times. We therefore augmented the training set by creating additional samples for every graph edge corresponding to the transition from v_0. We then perform training in a similar way to the standard object detector with the images with the ground truth bounding box and class labels (current state), while the previous state is being fused right after the YOLOv7 neck, as shown in Figure 3a. We do not modify the loss function of YOLOv7, which is the weighted sum of the box loss, object loss and class loss. During inference, just the image input and the previous state are needed.

In order to prevent overfitting we also apply dropout [31] to the state representation. We implement it by randomly skipping the *StateVec* module with a pre-defined probability for each training example. It forces the network into learning to utilize the visual information to the full extent in order to minimize the loss for the examples having their previous state suppressed.

3.4 Use of the detector in an AR application

Our overall goal is to integrate the state detector into an AR tutorial. To that end, we implemented an AR step-by-step tutorial application that runs on a HoloLens 2 headset, showing an interactive visualization of the state graph (Figure 1). The application is implemented using the Unity game engine. The headset client streams images over the local wireless network to a desktop computer, where the live state detection is running.

While the user steps through the tutorial, the next step is automatically highlighted by increasing its size and changing its position into focus. The next state is visualized either by pre-captured video

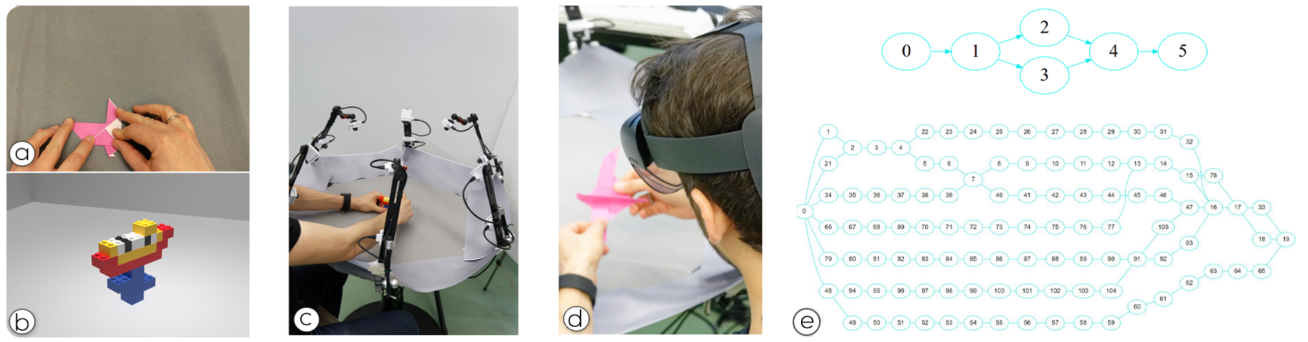

Figure 4: Datasets used in our work. We generate images and state graphs of several step-by-step-tutorials for training and testing our method. Samples are shown from the (a) Origami and (b) Lego flower synthetic datasets. We generate both synthetic and captured data. Synthetic data is automatically generated by rendering an object from several points of view while gradually disassembling it. We capture real data either using (c) a multi-camera rig, or (d) from a first-person perspective with the HoloLens 2 headset. (e) The data we use spans a range of complexities in terms of assembly graph, from 6 states for the Ikea table (top), to 106 states for the Engine block dataset (bottom).

snippets [40] or by images. If the user has transitioned to a new state in the assembly graph that is not part of the highlighted path, the visualization adapts to this choice and changes the path, showing the next instruction corresponding to what the user did.

Our method for detecting the current assembly state relies on the knowledge of the previous state. Since at runtime we need to confirm the state to update the visualization, we refine the prediction by a majority voting of the last predictions. The state with the highest number of votes is considered 'validated' if its number of votes exceeds a threshold. We set this threshold to the number of frames observed in one second. From the moment of validation onward, the previous state will be fed to the network for the next prediction, and the next state is displayed by the interface as a task for the user. The neural network is running on a desktop PC communicating with the headset over the local network. If the state has changed, the new state is sent to the headset client, and the visualization is updated.

4 Evaluation Data

A range of other datasets contain objects to be assembled. For example, the dataset introduced by Wang et al. [37] contains synthetic 3D models of IKEA furniture. Although this is a comprehensive resource for synthetic data, the parts are labeled separately. Ben-Shabat et al. [1] offer a real-world dataset of IKEA furniture where the labels consist of bounding boxes of the object parts, as well as segmentation masks for the parts. In both cases, manual labeling of states would be required to use state priors.

In contrast, in our work, the labels are object states, for example, an engine before and after a valve is removed. The states can be partially inferred from the presence of individual parts in the frame, but, in case the parts are not fully present and occluded, then they can be disambiguated only with additional knowledge. Liu et al. [14] use a state-labeled dataset comprised of two objects, a table and a fender object. Since the labels are object states, we use a part of this data in our work to evaluate our approach. We aim to also create our own data similar to this dataset, but with more complex objects and state graphs for evaluating our method.

The state graphs of the datasets that we use are not complete; they do not contain all the physically possible states, although our approach could be used with such graphs as well. We assume that the collected data results from some preferred or demonstrated ways of going through the states. Moreover, in practice, there usually exist limitations based on physical constraints, e.g., one part cannot be physically removed before another part. Nevertheless, we do assume that, from any given state, there can be a transition to multiple states, leading to a directed acyclic graph and not a simple list. State graphs could be created automatically through disassembly planning [13].

Table 1: Details about the datasets used for evaluation.

Dataset	Samples	States	Synthetic
Lego flower synthetic	1170	45	yes
Lego flower real	1296	45	no
Lego flower real extended	1557	45	no
Ikea table	773	6	no
Origami	4128	10	no
Engine block	8162	106	yes

For evaluating our method, we chose datasets of different origin. While the Ikea table dataset [14] represents furniture that is a common use case for AR assembly instructions, we also use Lego bricks for their high versatility. Furthermore, we use an origami dataset to demonstrate that our approach is not limited to rigid objects and a synthetic dataset showing CAD renderings of an engine block.

4.1 Captured datasets

Lego flower. To collect real data sets, we built a capture stage (Figure 4) consisting of a custom rig that holds a ring of 12 cameras observing the working area. The rig captures an assembly procedure simultaneously from many viewpoints. Our method does not depend on this kind of data capture, but using it significantly accelerates data capturing, especially when dealing with objects having a large number of states, since it captures multiple viewpoints, so a larger quantity of data at once.

We asked multiple test subjects to follow the AR tutorial for assembling the given object. Instructions are displayed on a tablet placed next to the capture stage, where users can manually navigate the written instructions, generated from 3D models with the Lic tool[1]. We vary the assembly sequences that we show to the users, in order to build branches in the state graphs of the objects from the recorded datasets. The Lego flower extended dataset just varies in terms of test set, that contains additional recordings.

The Lego set is highly configurable, allowing us to create and extend the state graph with multiple paths. Moreover, Lego bricks are easily confused. A conventional object detector might not be suitable to discriminate, for example, two 2×2 bricks of the same color placed next to each other from a single 2×4 brick of the same color, or an object configuration where the same part is present at multiple positions. Using state detection can help in disambiguation, since the state takes into consideration not only the presence, but

[1]www.bugeyedmonkeys.com/lic/

also the absence of a part, implicitly using the appearance of the part underneath to discriminate between states.

Origami. Additionally, we captured first-person recordings with a HoloLens 2 for our dataset of an Origami bird. The Origami use-case shows that we are not limited to assembly tasks or rigid objects. Another aspect that we consider in this dataset is the presence of hands. Origami is the only dataset that has the hands included, so the bounding boxes can contain the user's hand, leading to occlusions, as shown in Figure 4a.

For data labeling of our captured data, we use a rough bounding box proposal of the object, which we then manually check and refine. The initial bounding boxes are obtained either via color segmentation or by an object detector, GroundedDINO [15], working on descriptive text prompts. We then manually adjust incorrect boxes (imprecise, false negatives, false positives) and add the states as labels.

4.2 Synthetic datasets

Lego flower synthetic. For testing our detector under more controlled conditions, we generated several synthetic datasets in addition to the real ones. Synthetic datasets allow us to train on a large amount of data without having to perform manual annotation. We create a synthetic Lego dataset with the same states as the captured Lego flower using Mecabricks[2], LeoCAD[3] and Blender[4], followed by rendering synthetic frames in Blender. We use Python scripts to render a set of camera views for each state, with the camera being placed at viewpoints on a half-sphere around the object. Along with the images, we also render binary masks indicating where the object occupies the image, letting us automatically compute the objects' bounding boxes and perform automatic data annotation. A tutorial sequence is easily exported as a list referring to parts using the same names as in the Blender scene graph.

Engine block. Additionally, we created a synthetic dataset of an engine block [5] with more than 100 states. To create multiple datasets representing varying state graphs, we built a Unity application which lets the user interactively specify a disassembly sequence. Unlike the synthetic Lego flower, the engine block contains self-occluding parts. Including states where the important differences are occluded would confuse the network. Hence, we compute the visible parts and suppress the samples where no difference between the previous and next state is visible. This approach reflects what humans do when they turn the object until the relevant part comes into view.

5 Results

We tested our approach using our own captured and rendered data, and the data used in the approach by Liu et al. [14]. An overview of the data we use to evaluate our method can be found in Table 1. We compare to the original YOLOv7 detector as a baseline.

Each model is trained per object, where the object states are the classes. While never tested, the model can also be trained for multiple objects at once, but this may need additional considerations to define state transitions between different objects.

Setup. We evaluated our work on a desktop PC (GPU: NVIDIA RTX 4090, CPU: Intel Core i7 with 32 GB of RAM). Our implementation is an extension of the YOLOv7 codebase[6]. The inference runs on our hardware at an average of 33.7 ms/frame, i.e. 30 frames per second. For our experiments, we trained our detector, *StateYOLO*, each time for the same number of iterations with a batch size of 20 samples. We made sure to shuffle the data for each training procedure. For all our experiments we started with the weights provided with the YOLOv7 codebase, pre-trained on the COCO dataset, which we then fine-tuned on our own data. In our experiments, we use a drop-out rate of 0.5. We made sure to shuffle the data for each training procedure. For all our experiments we started with the weights provided with the YOLOv7 codebase, pre-trained on the COCO dataset, which we then fine-tuned on our own data. As data augmentation, we varied the image intensity, rotation and scale. Since the symmetry of the objects is important for object states, we did not use a flipping augmentation.

Table 2: Experiments on variants of the proposed architectures of our module on the Flower Synthetic dataset with and without dropout. The highest scores and higher scores than the original YOLOv7 are highlighted in bold fonts and underlines, respectively.

Architecture	**Dropout**	**P**	**R**	**Acc**	**mAP** 0.5	**mAP** 0.5:0.95
YOLOv7	-	0.850	0.940	0.792	0.906	0.906
StateYOLO-add	yes	0.848	**0.951**	$\underline{0.812}$	**0.94**	**0.94**
StateYOLO-mul	yes	0.836	0.925	0.777	0.896	0.896
StateYOLO-MLP-add	yes	$\underline{0.868}$	$\underline{0.943}$	$\underline{0.804}$	$\underline{0.917}$	$\underline{0.917}$
StateYOLO-MLP-mul	yes	0.830	0.917	0.757	0.887	0.886
StateYOLO-add	no	**0.890**	$\underline{0.942}$	**0.857**	$\underline{0.936}$	$\underline{0.936}$
StateYOLO-mul	no	0.783	0.906	0.729	0.882	0.87
StateYOLO-MLP-add	no	$\underline{0.871}$	0.936	$\underline{0.800}$	$\underline{0.917}$	$\underline{0.917}$
StateYOLO-MLP-mul	no	0.849	0.932	0.786	$\underline{0.907}$	$\underline{0.907}$

Performance measure. As a measure of performance, we computed the precision and recall scores as well as the mean average precision (mAP). The reason we choose to use these measures is that they are standard measures for object detection and are used in the papers introducing the YOLO object detector and its follow-up variants [28, 35]. This allows us to compare our state-enhanced method directly to the standard object detector. The difference is that in our case, the detected class is not the object itself, but variations of the object, meaning the states. Further, we use the accuracy measure, which also accounts for the true negatives to indicate how correct the model predictions are.

Precision, recall, and accuracy are measured at an intersection over union (IoU) threshold of 0.5. While the YOLOv7 framework computes the precision and recall at the maximum harmonic mean in regards to IoU, we chose to report the precision, recall, and accuracy at a fixed IoU value, which allows us to do the comparison in Section 5.3. We use a prediction confidence of 0.3. The two mAP scores in our tables refer to the IoU threshold: mAP 0.5 sets a fixed threshold of 0.5, while mAP 0.5:0.95 averages the score at 10 discrete IoU thresholds from 0.5 to 0.95 with a step of 0.05. This is a more robust measure of the performance of the object detector, since it considers the precision/recall curve over a range of IoU thresholds.

5.1 Module architecture

We investigated the performance of the module architectures described in Section 3.2. For this purpose we use the *Flower Synth* dataset, since it is generated within a controlled environment and also has a fairly large amount of states.

We compare the network architectures both with and without dropout training and present the results in Table 2. The highest mAP scores are obtained with the StateYOLO-add with dropout, so we chose this architecture to perform our further experiments. While StateYOLO-MLP-add also obtains a score better than the baseline, it seems that the multiplication operation in this case is

[2]www.mecabricks.com
[3]www.leocad.org
[4]www.blender.org
[5]https://grabcad.com/library/rotax582-c-1
[6]https://github.com/WongKinYiu/YOLOv7

Table 3: Performance of our method StateYOLO on various datasets, with different training techniques, on the testing sets of each data set. Details about the Ikea Table dataset can be found in the work by Liu et al. [14]. The higher scores are highlighted in bold font.

Dataset	**Method**	**Precision**	**Recall**	**Accuracy**	**mAP** 0.5	**mAP** 0.5:0.95
Lego Flower synthetic	YOLOv7	**0.85**	0.94	0.792	0.906	0.906
	StateYOLO-add dropout	0.848	**0.951**	**0.812**	**0.94**	**0.94**
Lego Flower real	YOLOv7	0.816	0.882	0.756	0.853	0.802
	StateYOLO-add dropout	**0.866**	**0.925**	**0.813**	**0.898**	**0.834**
Lego Flower real extended	YOLOv7	0.491	0.85	0.408	0.719	0.675
	StateYOLO-add dropout	**0.535**	**0.894**	**0.462**	**0.777**	**0.707**
Ikea table	YOLOv7	0.816	0.891	0.839	0.846	0.844
	StateYOLO-add dropout	**0.87**	**0.942**	**0.888**	**0.914**	**0.906**
Origami	YOLOv7	0.86	0.859	0.78	0.845	0.828
	StateYOLO-add dropout	**0.954**	**0.973**	**0.942**	**0.975**	**0.962**
Engine block	YOLOv7	0.743	0.818	0.625	0.777	0.777
	StateYOLO-add dropout	**0.839**	**0.928**	**0.785**	**0.902**	**0.902**

not an appropriate choice. The StateYOLO-MLP-mul architecture performs better when training without dropout, but still does not surpass the addition version.

5.2 Comparison with stateless network

To demonstrate our improved state detection over a network with no prior state information given, we compare our performance to that of the original YOLOv7. The results can be found in Table 3. The scores of our approach surpass the stateless method for all datasets in terms of precision, recall, accuracy, and mAP score. The results demonstrate that the performance increases with constraints by the given state priors.

The smallest difference in performance is achieved on the Flower datasets, where our method still obtains higher results than the baseline. This might be due to the fact that the states are easier to distinguish than in the case of the other datasets, a Lego part can be just present or absent. For the Ikea table, the states are defined in such a way that a leg that is not mounted fully straight affects the discrimination between two states.

The largest improvements are seen on the Engine block and on the Origami datasets. The Engine block is our dataset with the highest number of states. The results imply that our approach is particularly suitable for such a case.

5.3 Comparison with the Ikea table dataset

We evaluated our work on the Ikea table dataset introduced in the paper by Liu et al. [14]. Our results are not directly comparable to theirs, since they train on synthetic datasets and test on real datasets, while we only use the real dataset. We use a standard split (training 0.7, testing 0.2, validation 0.1). We achieve a better performance than Liu et al. [14] on the 'table' real dataset by a margin of around 5% in precision and 11% in recall at an IoU threshold of 0.5. These numbers should be interpreted with caution because of the different training set, and, also, because our test set is a random fraction of the captured dataset, while they test on the entire captured dataset.

5.4 Synthetic and recorded data

From our datasets, the flower object is the only one where real as well as synthetic data is available. Our results show that the improvement over the baseline is larger for the real dataset. It appears that YOLOv7 can properly discriminate fine details between states on the noise-free synthetic data, but not so well (compared to our enhanced version) for real data, where previous state information is more helpful.

5.5 Non-rigid objects and hand interactions

In order to evaluate our approach on a dataset showing non-rigid parts, we use the example of folding an Origami bird. This dataset contains states where the differences are more subtle, and the hands are also included in the bounding boxes.

We capture this dataset with three users, and then we split it into training, validation and testing. The results in Table 3 show that the detector can handle non-rigid objects well. Our approach improves the performance by around 13% mAP, and 16% on accuracy.

5.6 Further examples

The goal of our work is to create and study an approach that is usable in real-world scenarios. To this end, we also evaluate the performance of our detector on video recordings of users performing a step-by-step procedure (without AR guidance). By comparing our method to the original object detector, we identify cases where our method offers an advantage. For this qualitative evaluation, we used the Lego flower and the Origami bird. We recorded additional sequences following a written tutorial. The videos presented in this section were not used for training or quantitative evaluation.

We run our detector in the same fashion as described in Section 3, by keeping a cache of the last states, which indicate the majority vote for the next additional input to the network. Figure 5 shows examples of our method outperforming the baseline. In the case of the Lego flower assembly tutorial (in the top part of the figure), the user follows states #37, #38, and then #39. The baseline detector correctly identifies states #37 and #38, but confuses state #39 with state #45, since states #39 and #45 have a similar appearance. Given the information that the previous state was #38, our state-aware detector correctly predicts state #39.

The example with the Origami tutorial (Figure 5 bottom) shows the transition from state 5 to state 6. Also here, our detector distinguishes these correctly. The baseline detector initially detects state

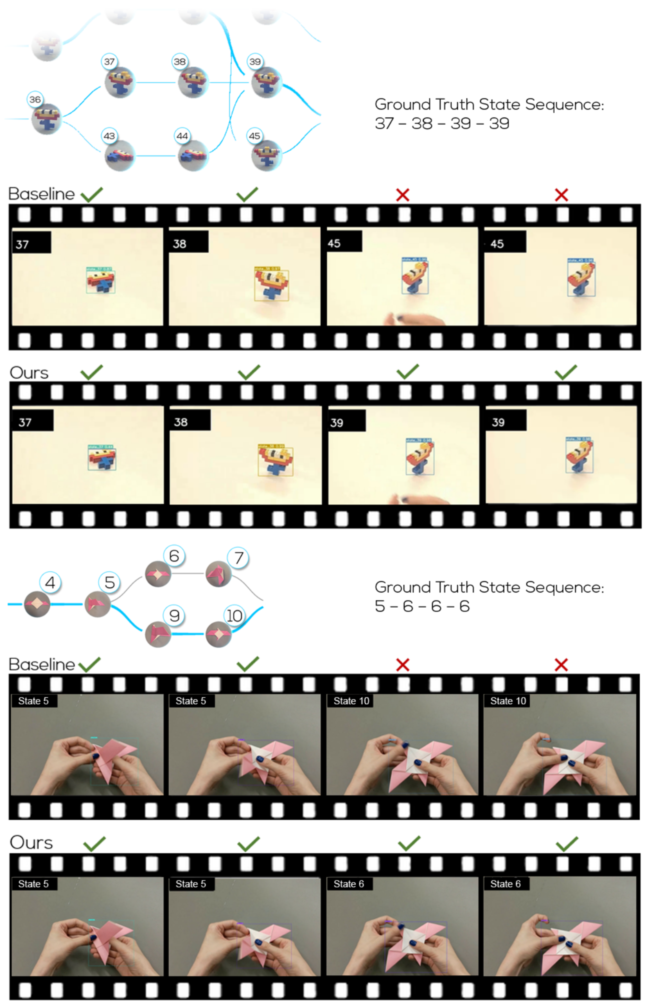

Figure 5: Example of different outputs on the baseline detector and our method. When states are visually similar, the baseline detector can end up in the wrong state. In contrast, given the contextual information of the previous state, ours predicts the correct state.

#6, then mistakenly state #10. This is because state #10 is similar in appearance to state #6, the difference being folding the bird's beak. As shown in the example, the previous state information here also helps distinguish this.

6 Discussion

Our results indicate that the contextual state information is beneficial, in the case when a state graph is available, however, the system also has a few limitations.

A trade-off of our method lies in the detection performance versus training time. While we achieve higher performance than a state of the art detector, training can take longer, because more data must be considered. Every distinct sample presented in the training phase corresponds to an edge in the state graph, so every image gives rise to many training samples, created by augmenting it with the possible previous states. Of course, some effort regarding the authoring of the state graph needs to be taken into account.

Furthermore, while our approach searches for the next state within the state graph, detecting unseen erroneous states is a current limitation, but would be interesting to explore in future work. Error detection could be modeled by our method by explicitly adding edges that represent frequent mistakes into the state graph. In this case, the edges must be labeled as 'correct' or 'incorrect'. Alternatively, at runtime, before validating the state, the state neighbors can be checked. The fast run-time allows for multiple predictions. If any of the neighbor transitions is predicted with a higher confidence, we can recover to that particular state.

We show that our method increases the performance over the baseline, but of course our enhanced detector is not guaranteed to always predict the correct state. To improve this, other information sources can be considered. Besides color images and state graphs, data regarding body tracking, hand tracking, gaze tracking or voice input could be fused into the network in a similar way like we do with the state information.

7 Conclusion and Future work

In the case of AR tutorials that do not require user validation, detecting the correct state is particularly important, since the tutorial cannot progress otherwise. Especially when states are similar in appearance, jumping to a wrongfully predicted position in the state graph can lead to an incorrect next instruction or the tutorial being stuck in a particular state. In such a case, improving the performance of an object detector by making it state-aware would positively impact the overall AR tutorial experience.

Motivated by our exploration of different architectures for the state-graph module shown in Table 2, more complex network architectures may also be considered when utilizing state graph priors [41]. We plan to explore how we can improve the performance of our network.

We explored heavy object occlusions using the origami example of a real video containing frames with hands covering the origami (Figure 4a), and the results demonstrate that our approach can handle such a case. Although we have not observed significant blurry images in our real datasets, investigating such edge cases would be interesting. We expect our method to also work better than the baseline with blurry images since the state graph provides additional information in comparison to a standard detector.

Another future direction is domain adaptation. For the scope of this paper, we use the same image type for training and testing, but, in practice, image types may vary. For instance, if a 3D model of the object is available, it can be used to render synthetic data, allowing training only on synthetic data or on a mixture of both real and rendered samples [14]. In this context, one could also explore different rendering techniques to increase realism or to simulate physical camera properties [19].

As a conclusion, we show that contextual information has the potential to improve state detection. We create three datasets for this task ranging from 10 to over 100 states to showcase the performance of our method, and also implement a proof of concept AR application using the enhanced detector. We designed our approach with a typical AR use case in mind – step-by-step tutorials – but the proposed method is not limited to this scenario. State-aware configuration detection can be employed wherever a fixed state graph is defined and a visual state validation is needed to progress to the next state, such as in industrial production. We make our datasets and code available on our project website, enabling further experiments and comparisons to follow-up work.

Acknowledgments

This work was supported by the Austrian Science Fund Lise Meitner (grant no. M3374) and FWF (grant no. P33634) and the Competence Center VRVis, which is funded by BMK, BMDW, Styria, SFG, Tyrol, and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (879730), which is managed by FFG. We want to express our gratitude to Professor Didier Stricker and Yongzhi Su for the shared datasets.

References

[1] Y. Ben-Shabat, X. Yu, F. Saleh, D. Campbell, C. Rodriguez-Opazo, H. Li, and S. Gould. The ikea asm dataset: Understanding people assembling furniture through actions, objects and pose. In *Proc. IEEE/CVF Winter Conf. on Applications of Computer Vision*, pp. 847–859, 2021. doi: 10.1109/WACV48630.2021.00089

[2] B. Bhattacharya and E. H. Winer. Augmented reality via expert demonstration authoring (AREDA). *Computers in Industry*, 105:61–79, 2019. doi: 10.1016/j.compind.2018.04.021

[3] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: Optimal speed and accuracy of object detection, 2020. doi: 10.48550/arXiv.2004.10934

[4] P. Breedveld. Observation, manipulation, and eye-hand coordination problems in minimally invasive surgery. In *Proc. European Conf. on Human Decision Making and Manual Control, Kassel*, pp. 219–231. Citeseer, 1997.

[5] S. Chidambaram, H. Huang, F. He, X. Qian, A. M. Villanueva, T. S. Redick, W. Stuerzlinger, and K. Ramani. Processar: An augmented reality-based tool to create in-situ procedural 2D/3D AR instructions. In *Designing Interactive Systems Conf.*, pp. 234–249, 2021. doi: 10.1145/3461778.3462126

[6] G. Chu, B. Potetz, W. Wang, A. Howard, Y. Song, F. Brucher, T. Leung, and H. Adam. Geo-aware networks for fine-grained recognition. In *Proc. the IEEE/CVF Int. Conf. on Computer Vision Workshops*, 2019. doi: 10.1109/ICCVW.2019.00033

[7] D. Damen, T. Leelasawassuk, O. Haines, A. Calway, and W. W. Mayol-Cuevas. You-do, i-learn: Discovering task relevant objects and their modes of interaction from multi-user egocentric video. In *Proc. British Machine Vision Conf. (BMVC)*, vol. 2, p. 3, 2014. doi: 10.5244/C.28.30

[8] N. Gavish, T. Gutiérrez, S. Webel, J. Rodríguez, M. Peveri, U. Bockholt, and F. Tecchia. Evaluating virtual reality and augmented reality training for industrial maintenance and assembly tasks. *Interactive Learning Environments*, 23(6):778–798, 2015. doi: 10.1080/10494820.2013.815221

[9] R. Girshick. Fast r-cnn. In *Proc. Int. Conf. on Computer Vision (ICCV)*, pp. 1440–1448, 2015. doi: 10.1109/ICCV.2015.169

[10] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 580–587. IEEE Computer Society, 2014. doi: 10.1109/CVPR.2014.81

[11] A. Gupta, D. Fox, B. Curless, and M. Cohen. Duplotrack: a real-time system for authoring and guiding duplo block assembly. In *Proc. ACM Symp. on User Interface Software and Technology (UIST)*, pp. 389–402, 2012. doi: 10.1145/2380116.2380167

[12] S. J. Henderson and S. K. Feiner. Augmented reality in the psychomotor phase of a procedural task. In *Proc. Int. Symp. on Mixed and Augmented Reality (ISMAR)*, pp. 191–200. IEEE, 2011. doi: 10.1109/ISMAR.2011.6092386

[13] B. Kerbl, D. Kalkofen, M. Steinberger, and D. Schmalstieg. Interactive disassembly planning for complex objects. *Computer Graphics Forum*, 34(2):287–297, may 2015. doi: 10.1111/cgf.12560

[14] H. Liu, Y. Su, J. Rambach, A. Pagani, and D. Stricker. Tga: Two-level group attention for assembly state detection. In *IEEE Int. Symp. on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pp. 258–263. IEEE, 2020. doi: 10.1109/ISMAR-Adjunct51615.2020.00074

[15] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, J. Yang, H. Su, J. Zhu, and L. Zhang. Grounding dino: Marrying dino with grounded pre-training for open-set object detection, 2023. doi: 10.48550/arXiv.2303.05499

[16] Z. Liu, Z. Zhu, E. Jiang, F. Huang, A. M. Villanueva, X. Qian, T. Wang, and K. Ramani. Instrumentar: Auto-generation of augmented reality tutorials for operating digital instruments through recording embodied demonstration. In *Proc. ACM Conf. on Human Factors in Computing Systems (CHI)*, pp. 1–17, 2023. doi: 10.1145/3544548.3581442

[17] O. Mac Aodha, E. Cole, and P. Perona. Presence-only geographical priors for fine-grained image classification. In *Proc. Int. Conf. on Computer Vision (ICCV)*, pp. 9596–9606, 2019. doi: 10.1109/ICCV.2019.00969

[18] S. Makris, G. Pintzos, L. Rentzos, and G. Chryssolouris. Assembly support using AR technology based on automatic sequence generation. *CIRP Annals*, 62(1):9–12, 2013. doi: 10.1016/j.cirp.2013.03.095

[19] D. Mandl, P. M. Roth, T. Langlotz, C. Ebner, S. Mori, S. Zollmann, P. Mohr, and D. Kalkofen. Neural cameras: Learning camera characteristics for coherent mixed reality rendering. In *Proc. Int. Symp. on Mixed and Augmented Reality (ISMAR)*, pp. 508–516. IEEE, 2021. doi: 10.1109/ISMAR52148.2021.00068

[20] A. Miller, B. White, E. Charbonneau, Z. Kanzler, and J. J. LaViola Jr. Interactive 3d model acquisition and tracking of building block structures. *IEEE Trans. on Visualization and Computer Graphics (TVCG)*, 18(4):651–659, 2012. doi: 10.1109/TVCG.2012.48

[21] P. Mohr, B. Kerbl, M. Donoser, D. Schmalstieg, and D. Kalkofen. Retargeting technical documentation to augmented reality. In *Proc. ACM Conf. on Human Factors in Computing Systems (CHI)*, pp. 3337–3346, 2015. doi: 10.1145/2702123.2702490

[22] P. Mohr, D. Mandl, M. Tatzgern, E. Veas, D. Schmalstieg, and D. Kalkofen. Retargeting video tutorials showing tools with surface contact to augmented reality. In *Proc. ACM Conf. on Human Factors in Computing Systems (CHI)*, pp. 6547–6558, 2017. doi: 10.1145/3025453.3025688

[23] P. Mohr, S. Mori, T. Langlotz, B. H. Thomas, D. Schmalstieg, and D. Kalkofen. Mixed reality light fields for interactive remote assistance. In *Proc. ACM Conf. on Human Factors in Computing Systems (CHI)*, pp. 1–12, 2020. doi: 10.1145/3313831.3376289

[24] N. Petersen, A. Pagani, and D. Stricker. Real-time modeling and tracking manual workflows from first-person vision. In *Proc. Int. Symp. on Mixed and Augmented Reality (ISMAR)*, pp. 117–124. IEEE, 2013. doi: 10.1109/ISMAR.2013.6671771

[25] N. Petersen and D. Stricker. Learning task structure from video examples for workflow tracking and authoring. In *Proc. Int. Symp. on Mixed and Augmented Reality (ISMAR)*, pp. 237–246. IEEE, 2012. doi: 10.1109/ISMAR.2012.6402562

[26] S. Pongnumkul, M. Dontcheva, W. Li, J. Wang, L. Bourdev, S. Avidan, and M. F. Cohen. Pause-and-play: automatically linking screencast video tutorials with applications. In *Proc. ACM Symp. on User Interface Software and Technology (UIST)*, pp. 135–144, 2011. doi: 10.1145/2047196.2047213

[27] X. Qian, F. He, X. Hu, T. Wang, A. Ipsita, and K. Ramani. Scalar: Authoring semantically adaptive augmented reality experiences in virtual reality. In *Proc. ACM Conf. on Human Factors in Computing Systems (CHI)*, CHI '22. Association for Computing Machinery, New York, NY, USA, 2022. doi: 10.1145/3491102.3517665

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016. doi: 10.1109/CVPR.2016.91

[29] D. Reiners, D. Stricker, G. Klinker, and S. Müller. Augmented reality for construction tasks: Doorlock assembly. In *Proc. Int. Workshop on AR: Placing artificial objects in real scenes*, pp. 31–46. AK Peters, Ltd., 1999.

[30] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 39(6):1137–1149, 2017. doi: 10.1109/TPAMI.2016.2577031

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[32] A. Stanescu, P. Mohr, D. Schmalstieg, and D. Kalkofen. Model-free authoring by demonstration of assembly instructions in augmented reality. *IEEE Trans. on Visualization and Computer Graphics (TVCG)*, 28(11):3821–3831, 2022. doi: 10.1109/TVCG.2022.3203104

[33] K. Tang, M. Paluri, L. Fei-Fei, R. Fergus, and L. Bourdev. Improving image classification with location context. In *Proc. Int. Conf. on Computer Vision (ICCV)*, pp. 1008–1016, 2015. doi: 10.1109/ICCV.2015.121

[34] B. Wang, G. Wang, A. Sharf, Y. Li, F. Zhong, X. Qin, D. CohenOr, and B. Chen. Active assembly guidance with online video parsing. In *Proc. IEEE Virtual Reality (VR)*, pp. 459–466. IEEE, 2018. doi: 10.1109/VR.2018.8446602

[35] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 7464–7475, June 2023.

[36] R. Wang, Y. Zhang, J. Mao, C.-Y. Cheng, and J. Wu. Translating a visual lego manual to a machine-executable plan. In *Proc. European Conf. on Computer Vision (ECCV)*, pp. 677–694. Springer, 2022. doi: 10.1007/978-3-031-19836-6_38

[37] R. Wang, Y. Zhang, J. Mao, R. Zhang, C.-Y. Cheng, and J. Wu. Ikea-manual: Seeing shape assembly step by step. *Advances in Neural Information Processing Systems*, 35:28428–28440, 2022.

[38] M. Whitlock, G. Fitzmaurice, T. Grossman, and J. Matejka. AuthAR: Concurrent authoring of tutorials for AR assembly guidance. In *Proc. Graphics Interface*, p. 431 – 439, 2020. doi: doi.org/10.20380/GI2020.43

[39] L.-C. Wu, I.-C. Lin, and M.-H. Tsai. Augmented reality instruction for object assembly based on markerless tracking. In *Proceedings ACM Symposium on Interactive 3D Graphics and Games*, pp. 95–102, 2016. doi: 10.1145/2856400.2856416

[40] M. Yamaguchi, S. Mori, P. Mohr, M. Tatzgern, A. Stanescu, H. Saito, and D. Kalkofen. Video-annotated augmented reality assembly tutorials. In *Proc. ACM Symp. on User Interface Software and Technology (UIST)*, pp. 1010–1022, 2020. doi: 10.1145/3379337.3415819

[41] L. Yang, X. Li, R. Song, B. Zhao, J. Tao, S. Zhou, J. Liang, and J. Yang. Dynamic mlp for fine-grained image classification by leveraging geographical and temporal information. In *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 10945–10954, 2022. doi: 10.1109/CVPR52688.2022.01067

[42] J. Zauner, M. Haller, A. Brandl, and W. Hartman. Authoring of a mixed reality assembly instructor for hierarchical structures. In *Proc. Int. Symp. on Mixed and Augmented Reality (ISMAR)*, pp. 237–246, 2003. doi: 10.1109/ISMAR.2003.1240707

[43] B. Zhou and S. Güven. Fine-grained visual recognition in mobile augmented reality for technical support. *IEEE Trans. on Visualization and Computer Graphics (TVCG)*, 26(12):3514–3523, 2020. doi: 10.1109/TVCG.2020.3023635

[44] V. Zogopoulos, E. Geurts, D. Gors, and S. Kauffmann. Authoring tool for automatic generation of augmented reality instruction sequence for manual operations. *Procedia CIRP*, 106:84–89, 2022. doi: 10.1016/j.procir.2022.02.159