IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 8, 2022, accepted January 29, 2022, date of publication February 7, 2022, date of current version February 15, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3149713

An End-to-End Stochastic Action and Visual Estimation System Towards Autonomous Teleoperation

ABDULLAH AKAY AND YUSUF SINAN AKGUL
Department of Computer Engineering, Gebze Technical University, 41400 Kocaeli, Turkey
Corresponding author: Abdullah Akay (aakay@gtu.edu.tr)

This work was supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under Project 112E127.

ABSTRACT Teleoperation systems have been getting significant attention from many application areas for decades. However, classical teleoperation systems suffer from problems such as lack of natural feedback, latency, and inefficient operator throughput. Researchers attempted to address these issues by performing some of the teleoperation sub-tasks autonomously whenever requested by the operator. Nevertheless, these systems still need the operator to see the need for autonomous actions and initiate these actions manually, which is demanding for the operators. This paper proposes a novel end-to-end Stochastic Assistive Teleoperation System (SATS) that always stays in the loop, automatically detects applicable actions with probabilities, and produces visual scene estimations for each of these actions, which results in increased operator efficiency and throughput. We introduce several methods that combine ideas from Markov processes and recurrent neural networks to stochastically predict future action sequences and scene configurations with tractable algorithms. Experiments performed with a group of operators on real and simulative teleoperation environments show that operators issue a considerably smaller number of commands compared to alternative methods. We also showed that the operators can manipulate multiple robots simultaneously using our technique, which boosts the operator throughput even further. We provide supplementary video material that demonstrates SATS in action.

INDEX TERMS Action-conditioned prediction, assistive teleoperation, Markov chains, recurrent neural networks, semantic video segmentation, stochastic video prediction.

I. INTRODUCTION

A typical teleoperation system is composed of a human operator maneuvering a slave robot working on a remote or hard to reach environment [1]–[4]. Motivated by the large variety of applications, ranging from space explorations [2], [5] and deep underwater explorations [6], [7] to UAV operations [8], [9] and nuclear waste decontamination [10], [11], teleoperation have been studied extensively. However, some of the fundamental problems of teleoperation remain to be solved. For example, the lack of natural haptic feedback and data transfer latency between the remote system and the operator are some of these problems that cause inefficiencies for the operators. Besides, humans are prone to errors in environments where repetitive tasks have to be handled often, which is the case for many teleoperation environments [12]–[14]. In order to address these problems, autonomous teleoperation systems are often proposed for missions in remote environments such as deep sea [6], [7], [14], air [8], [9], space [15], nuclear disaster sites [11], industrial sites [16], [17] or for any generic site [18]–[23]. Many of these autonomous tasks mainly involve basic processes like grasping [16], rotating [19] or dragging [17] an object, opening a valve [7], or hot-stabbing [6]. The operators of these autonomous systems usually perform their tasks manually, and they trigger the autonomous system when it is needed. In other words, the overall teleoperation process is not monitored by the autonomous system; the initiation of the autonomy is supposed to be done by the operators, which brings extra mental loads to the operators.

In this paper, we propose a Stochastic Assistive Teleoperation System (SATS) that continuously monitors the complete teleoperation process, including the operator actions, robotic arm, and the remote site configurations. During this

The associate editor coordinating the review of this manuscript and approving it for publication was Aysegul Ucar.

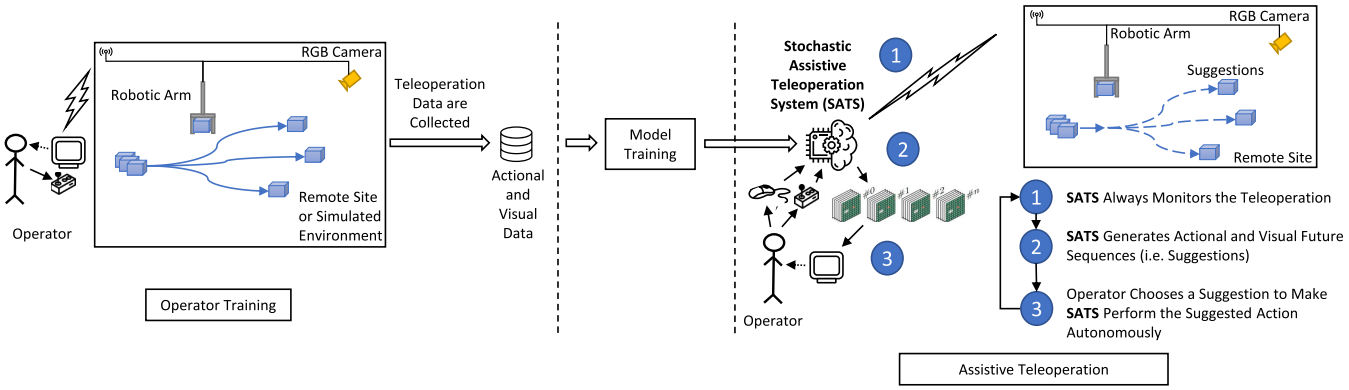


FIGURE 1. **Overall structure of SATS. Left. SATS is trained using actional and visual data obtained while the operator is trained. Right. During regular operation, SATS always stays in teleoperation loop and continuously makes action sequence suggestions. Each suggestion includes visual scene estimations with probabilities.**

continuous monitoring process, when SATS automatically detects autonomously applicable actions, it offers these actions to the operators as a list sorted by the likelihood of the action applicability. Different from the classical autonomous teleoperation systems, the SATS operators also see the estimated visual outcomes of these actions if applied. The visual scene estimations, along with their probabilities, make it more convenient to choose a selection out of all possible actions. SATS is an end-to-end smart teleoperation system because it is always in the teleoperation loop and ready for making action suggestions. Therefore, in contrast to classical autonomous systems employing learning from demonstration techniques [7], [14], SATS does not require any triggering from the operators for generating suggestions.

For practical teleoperation systems, operator training is usually performed in simulative environments that are quite similar to their real counterpart in robotic system dynamics, and environmental appearance [24]. Similarly, SATS can employ a simulative environment for operator training in which operators can control a robotic arm to perform various tasks (Fig. 1, left). We define a set of teleoperation scenarios and hold training sessions to train the operators for these scenarios. SATS collects a large amount of actional and visual data during these sessions. Actional data are obtained from robotic commands issued by the operators, and visual data are captured from simulative or real RGB cameras placed on fixed positions at the teleoperation site. We train the SATS model using the data collected (Fig. 1, center). SATS deploys the trained model to make it observe the teleoperation process while the operators perform their tasks by controlling the robot manually (step 1 in Fig. 1, right). After observing the environment for a short time, the model generates multiple actional and visual predictions (step 2 in Fig. 1, right). The generated suggestions sorted with respect to their likelihoods are shown to the operators on the screen as a list. If the operators find a suggestion useful, they just choose that suggestion to activate the autonomous mode and make SATS perform the suggested action sequence autonomously (step 3 in Fig. 1, right). SATS keeps monitoring the scene while it performs the assigned task. Therefore, it is always ready to generate further predictions at any time during the teleoperation. Note that the operators stay free while SATS executes the sequence of actions chosen by the operator. SATS is designed in a way that the operator can use this free time to control another available robot. As a result, the throughput of the operators increases considerably, making our assistive system potentially very efficient for the teleoperation missions.

The defined problem is an example of stochastic sequence prediction tasks in which sequences of observations are emitted from a set of unknown hidden states. One of the popular ways of solving these kinds of problems is to employ Hidden Markov Models (HMM), where observations correspond to captured RGB frames, and hidden states correspond to high-level information about the scene such as semantic segmentations of the scene objects or events of the scene [25]. Although HMM's are attractive for our task due to their inherent stochastic nature and strong theoretical foundations, using HMMs for a complex task like ours is intractable, as will be discussed in detail in the following sections. Hence, we introduce a technique incorporating deep neural networks and Markov chains in a unique way to offer a feasible and efficient solution to our stochastic sequence prediction problem.

Recently, the video prediction community introduced various approaches to predict future video frames for a given sequence of video frames [17], [26]–[28], which is conceptually similar to our problem of visual prediction. However, many of these methods do not consider the case where multiple alternative future frame sequences are possible. Most of the time, the generated video frames turn out to be the average of all possible future frames [17], which makes them blurry and unusable for our problem. There are several future frame prediction methods [27], [28] that produce multiple alternative future frames. However, the produced alternative video frames are random, and they may miss many likely

cases, which would be very limiting for our purposes. Unlike these video prediction approaches, the proposed method predicts future frames in a stochastic way, which generates all the possible future frame sequences with actions and their probability distributions at once as a list at each time-step.

We provide thorough experiments to validate our system. The experimental results verify that the operators issue a much smaller number of commands compared to the alternative methods. It is shown that visual cues provided by SATS facilitate decision making process of operators when there are multiple possible future sequences. We demonstrated that SATS significantly boosts the throughput of the operators by enabling them to use multiple robots. Finally, we showed that the proposed method can be trained on synthetic data, which is obtained from simulative environments and can be deployed in real environments.

Our main contribution in this study is developing an AI-powered assistive teleoperation system that learns task patterns in an unsupervised way, makes multiple actional and visual estimations continuously during teleoperation, and performs autonomous teleoperation whenever possible. It is important to emphasize that our teleoperation system does not have all the features of the traditional teleoperation systems. We abstracted the basic teleoperation sub-tasks such as gripping and releasing because we focused on managing teleoperation missions at higher levels as a first step. With that being said, we believe that our framework is sufficiently sophisticated to show the potential applicability of SATS. We will further explain this point throughout the paper. Our contributions in this study can be listed as follows:

- An end-to-end assistive teleoperation framework that always stays in the loop and detects applicable actions automatically;
- A novel incorporation of deep neural networks and Markov chains to efficiently generate multiple future sequence predictions along with their probability distributions;
- Stochastic actional and visual future predictions that are shown to boost operator efficiency significantly;
- Increased operator throughput by allowing operators to control multiple robots simultaneously while the system performs autonomous tasks.

The rest of the paper is organized as follows: in Section II, related work is investigated. The design steps of the proposed method are described in Section III. In Section IV, the real and the simulative 3D environments, as well as corresponding datasets, are introduced. In the same section, testing procedures of the proposed system are described, and the experimental results are shown. Finally, in Section V, we provide concluding remarks.

II. RELATED WORK

Since our contributions are mainly in the fields of assistive teleoperation and action or video prediction, we set our focus on recent work in these fields.

A. SEMI-AUTONOMOUS/ASSISTIVE/SHARED TELEOPERATION

There is a strong demand for automated and smart teleoperation systems in the community. [29], which is our prior work, introduced a solution for scene analysis in assistive teleoperation by analyzing 3D reconstruction of the teleoperation scene and extracting segment sequences. This work is based on learning from demonstration, which requires manually extracted samples of each teleoperation task. [30] presented another learning from demonstration technique to automate industrial robots using deep learning models. They used a surgical da Vinci robot to manipulate objects in the teleoperation scene. [31] proposed a semi-autonomous teleoperation technique that also can learn from demonstration. They used Gaussian Mixture Models (GMM) to build the proposed assistance system, which reduces the workload of the operators. Similarly, [32] proposed a semi-autonomous teleoperation framework that contains robots that are capable of learning from previously performed tasks. [6], [7], [14] are focused on semi-autonomous teleoperation systems based on HMM models that work on deep-sea environment to complete tasks such as opening a valve via a robotic arm. Although many other similar techniques, such as [18]–[21], proposed various levels of autonomy in teleoperation, their techniques are mainly used for performing only local and basic tasks autonomously.

[33] and [34] proposed deep models for mobile robots that can be used to estimate future traversability. Their models generate future robot velocities and video frames to autonomously control the robot and give an idea about the consequences of the autonomous control to operators. The main focus of these methods is autonomous collision avoidance in robot navigation. Their method requires a manual input point on the image space of the robot camera, indicating the target point to initialize autonomous control.

The methods described so far are mainly based on learning from demonstration technique, which has significant restrictions on building sophisticated models. First, each sample of action sequence is needed to be manually labeled during the data collection process, which is costly in terms of experienced operator time. In addition, the resulting model should be triggered manually whenever it is needed for each desired action type during the real operation. Conversely, our data collection process is transparent to the operators. We do not force the operators to perform actions just for collecting data. Operators only try to train themselves to get familiar with the teleoperation environment. We record all the raw data observed during the operator training process and train our system with the collected data. Furthermore, operators do not have to manually trigger the autonomous control for a specific action type in our system. SATS always monitors the environment and automatically detects possible applicable actions in real-time.

The reason we choose the term assistive teleoperation is that our system is not designed for helping the operators in

low-level subtasks but designed for supporting the operators at higher levels as a generative decision support system. The proposed system generates multiple predictions about probable future sequences in terms of actional and visual aspects and offers them to the operators as decision suggestions which makes the proposed method a high-level assistive system. In contrast, a typical shared-control system is more of a low-level assistive system which is useful in simpler tasks, i.e., assisting operators when approaching a target object by decreasing the DoF of the robot [35]–[37]. Similarly, semi-autonomous teleoperation term is not sufficient to describe our work, as it is commonly used for learning from demonstration systems. Please note that those systems are inferior to the proposed method due to the reasons we have mentioned previously. Finally, we did not use the term autonomous teleoperation to describe our work as our system is not fully autonomous in the sense that it still is supposed to work with an operator. In conclusion, the broadest term that describes AI-powered high-level supportive teleoperation systems is assistive teleoperation.

B. ACTION AND VIDEO PREDICTION

SATS, in part, is conceptually similar to state-of-the-art video or action-conditioned video prediction systems. We briefly describe these systems and explain the differences from the proposed method. [38] proposed a convolutional network that can generate future video frames from a given input sequence. Particularly, they investigate how choosing different types of loss functions affects the quality of the frame predictions. [39] proposed a deep learning framework based on a Generative Adversarial Network (GAN) for video prediction and completion in human action videos. Both [38] and [39] predict future video sequences without considering user actions. There are other methods that take the user actions and previous video frames into consideration during the future frame prediction, which is called action-conditioned future frame estimation. For example, [26] introduced a deep architecture for predicting the future video frames that are dependent on both previous frames and agent actions in Atari games. They collected 500000 frames using DeepMind's Deep-Q-Network for Atari Games to train their model. They showed that the method is able to predict realistic future frames on the Atari game domain. They focused on only Atari games, and the applicability of their method on real videos remained unclear. Similarly, [40] suggested an action-conditioned video prediction model that incorporates appearance information from previous frames and robot instructions. They reported that the model can produce satisfactory results for more than 10 time-steps into the future. [17] presented a learning-based approach for basic robot manipulation, which is trained using 50000 randomly generated object pushing attempts. They use a deep predictive neural network model to plan actions that should be taken by the robot. Their method generates an action sequence that can be used by the robot to achieve the desired goal. Unlike our method, their technique takes the source and target locations as input and uses that information in the objective function during the operation. Both [40] and [17] focus only on basic tasks like pushing or dragging objects on the scene. Another action-conditioned video prediction system is by [41] that introduced a light-weight video prediction technique that can predict up to 10 frames in the future in a LEGO Mindstorms robot environment. Their method can generate only a single visual future sequence at a time-step. While the idea of future video frame prediction of the above studies is similar to our method, they do not take into consideration the case of possible multiple future sequences.

[27] focused on a stochastic variational video prediction model and developed a deep model that predicts different future video sequences employing latent variables. [28] proposed a model based on Variational Auto Encoder-GANs to achieve stochastic video prediction. The main difference between [27], [28], and our method is that our system generates a video sequence distribution of future video frames. In other words, each predicted video sequence is assigned a probability value that would be used for prioritizing these suggestions while offering them to the operator. Both [27] and [28] produce alternative future frames by sampling a special latent random variable, which produces no information about the occurrence probabilities of the sequences. Please note that none of the action and video prediction methods use assistive teleoperation as a target application. To the best of our knowledge, our system is the first to use action conditioned video prediction methods for the task of assistive teleoperation.

III. THE METHOD

We define a number of action categories $A = [G, F, B, L, R, D, P, N]$ each of which stands for ***G**rip the object, move **F**orward, move **B**ackward, move **L**eft, move **R**ight, **D**rop the object, rotate the object **P**ositive 90 degrees around Y axis* and *rotate the object **N**egative 90 degrees around Y axis*, respectively. Formally, we define an operator action at time-step i as a one-hot encoded vector $\mathbf{a}_i \in \{0, 1\}^n$, where $n = |A|$. The hot element of $\mathbf{a}_i$ indicates the index of the corresponding action category defined in A. For example, $\mathbf{a}_k = [0, 1, 0, 0, 0, 0, 0, 0]$ means the user issues a **F**orward command at time-step k. There must be exactly one action for each of the video frames or time-steps. We omit time-steps that do not contain any actional information. A sequence $\mathbf{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_{c-1}]$ is defined as a vector of action frame pairs, where $\mathbf{x}_i = [\mathbf{a}_i, \mathbf{f}_i]$. The video frame $\mathbf{f}_i \in \mathbb{R}^2$ contains visual information about the scene. Note that, although our formulations consider only a single video frame for each time-step, it is trivial to extend this model with multiple frames from multiple cameras. Given an input sequence $\mathbf{X}$, our goal is to predict a distribution of future sequences $\mathbb{Y} = [\mathbf{Y}_0, \ldots, \mathbf{Y}_{n-1}]$. The elements of $\mathbb{Y}$ represent an alternative future sequence ordered by their estimated probabilities, which sum up to 1.0 as expected. Elements of a predicted future sequence $\mathbf{Y}_i = \left[[\mathbf{a}_0^i, \mathbf{f}_0^i], \ldots, [\mathbf{a}_{L-1}^i, \mathbf{f}_{L-1}^i]\right]$ contains

predicted actions ($\mathbf{a}_j^i$) and the corresponding Visual Semantic Segment Information (VSSI) ($\mathbf{f}_j^i$) for each time-step, where L denotes the length of the predicted future sequence. The estimated VSSI images are the same size as the input images, and they assign a semantic segment label for each pixel of the scene. These labels are *ForeGround* (FG, objects manipulated by the robotic arm), *Robotic Arm* (RA, the pixels that correspond to the arm itself), and *BackGround* (BG, any scene element other than objects and the robotic arm). We use VSSI images mainly for displaying the predicted future scenes to the operator.

A. CONVENTIONAL HIDDEN MARKOV MODEL APPROACH

In order to provide a solution to the described estimation problem, one can propose employing HMMs due to their inherent stochastic system state estimation capabilities. HMMs can also relate the estimated system states between the time-steps, which would incorporate crucial temporal information during the future prediction process. However, as we will show below, the standard employment of HMMs is problematic for our specific problem due to computational tractability issues.

An HMM of order N is defined by $\lambda = (T, O)$, where T denotes transition probability matrix, and O represents observation likelihoods. The hidden states of HMM describe a combination of operator actions and VSSI of the teleoperation scenes. For the sake of explanation convenience, let us consider a simplified example in which λ takes images containing only a single pixel along with regular operator actions. In such a case, the hidden system state table contains $3|A| = H$ states, which is a combination of 3 VSSI states and $|A| = 8$ operator actions. The best hidden state sequence of length 1 explaining the observed operator action-image combination sequence can be estimated easily using the standard HMM decoding methods because the state size of 24 is manageable for this case. However, if we extend the length of the prediction sequence to L, we need H^L hidden states to represent all possible action-VSSI combinations. Obviously, the number of hidden states in λ quickly becomes unmanageable to be implemented for practical values of L (e.g., 24^{20} for predicting 20 frames in the future). Note that this is the simplified case for only one-pixel input images. If we consider the general case for $w \times h$ input image sizes, the number of total required hidden states becomes $(whH)^L$, which is intractable. Finally, the standard decoding methods of HMMs would produce only the best estimated sequence. However, our problem definition requires estimation of many alternative sequences to be presented to the operator. Although HMMs are powerful due to their stochastic nature and ability to relate adjacent time-steps, they are not directly applicable to our problem. As a result, we need to develop novel methods to stay tractable while keeping desirable features of HMMs.

B. OUR HYBRID TECHNIQUE

We propose a new technique as a feasible alternative to the HMM approach above. We incorporate a deep recurrent neural network $\mathcal{N}$ and a special version of first-order Markov chain $\mathcal{C} = (\mathrm{T})$, where T denotes the state transition tensor to replace HMM λ. By employing this approach, we come up with a manageable and efficient future sequence prediction system. For the sake of explainability, we describe the proposed solution using the simplified one-pixel image case as we did with HMMs. $\mathcal{C}$ has a set of $3|A| = H$ states, each denoting Action-Conditioned Semantic Segmentation (ACSS) labels. Basically, ACSS defines $H = 24$ labels representing all VSSI-operator action combinations such as $[BG - F, \ldots, BG - D, RA - F, \ldots, RA - D, FG - F, \ldots, FG - D]$. The state transition probabilities of $\mathcal{C}$ are defined slightly differently from the traditional Markov transitions. Conventionally, state transition probability matrix T is an $H \times H$ matrix whose values are learned during the training phase, and these values do not change during a decoding process. In contrast, we define a dynamic state transition tensor $\hat{\mathbb{T}}$ as an $L \times H \times H$ to allow each state transition probability to take on different values at each iteration in the decoding process. In other words, the state transition matrix T of $\mathcal{C}$ is not constant and can take different values for each future time-step t_j during the decoding process. To do so, we train the neural network $\mathcal{N}$ to learn to predict a state transition probability matrix $\hat{\mathbb{T}} = [\hat{\mathrm{T}}_0, \ldots, \hat{\mathrm{T}}_L]$ for each future time-step t_j using the input sequence of operator actions and images, which produces L transition matrices in total (Fig. 2). We stack all the predicted transition matrices $\hat{\mathrm{T}}_j$ into a 3D tensor $\hat{\mathbb{T}}$, which becomes the transition parameter of the Markov chain $\mathcal{C} = (\hat{\mathbb{T}})$. $\mathcal{C}$ uses j^{th} transition probability matrix $\hat{\mathrm{T}}_j$ at j^{th} iteration in the decoding process. Decoding (sampling) a future ACSS sequence of length L can be done by just iterating L times through $\mathcal{C}$ and its sequence probability can be computed by calculating joint probabilities (i.e., transition probabilities) of visited states. In this way, we simulate a Markov chain of order N using a first-order Markov chain.

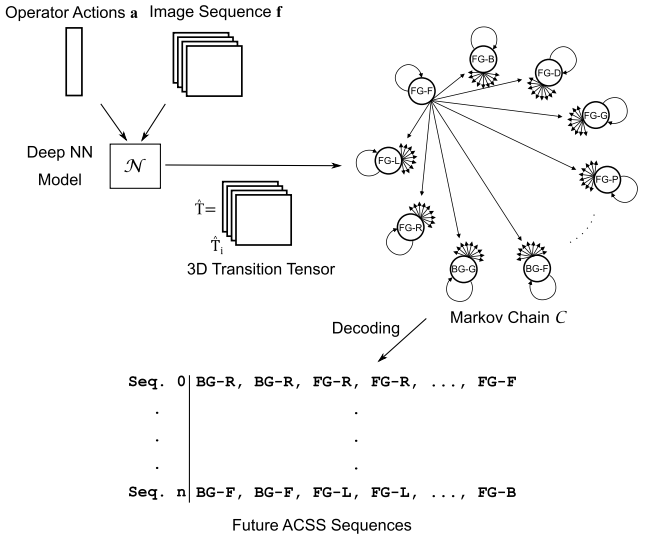


FIGURE 2. Future sequence generation for one-pixel image using our heuristic method.

As an example, let us generate a future ACSS for the simplified one-pixel-image example. First, $\mathcal{N}$ takes the input sequence and generates $\mathcal{C}_c$ at time-step t_c. Then, we iterate L times through $\mathcal{C}_c$ and emit an ACSS label at each iteration of the decoding process. After we finish the iterations, we have a future ACSS sequence of length L for the only pixel of the input image (e.g. [FG-F, FG-F, FG-R, ..., RA-D]).

In the general case, where input is a sequence of operator actions and images of size $w \times h$, $\mathcal{N}$ samples $w \times h$ matrix of Markov chains $\boldsymbol{\mathcal{C}}_c$ at time-step t_c. Elements of the Markov chain matrix are defined as $\boldsymbol{\mathcal{C}}_{c,k,l}$. Each element of this matrix samples a future ACSS sequence for a pixel coordinate p_{kl} starting from time-step t_c. The generated ACSS sequence for each pixel coordinate is defined as $\hat{\mathbf{S}}_{c,k,l} = dec(\boldsymbol{\mathcal{C}}_{c,k,l}(\hat{\mathbf{T}}_{c,k,l}))$, where $\hat{\mathbf{T}}$ is a 3D tensor of state transition tensors $\hat{\mathrm{T}}$ and dec is the decoding function of the Markov chains. Each $\hat{\mathbf{T}}_{c,k,l}$, which is sampled at time-step t_c, corresponds to a 3D transition tensor $\hat{\mathrm{T}}$ for pixel coordinate p_{kl}. Since we need top n best ACSS sequences, we define a set of future ACSS sequences $\hat{\mathbf{S}}_{:,c,k,l} = [\hat{\mathbf{S}}_{0,c,k,l}, \ldots, \hat{\mathbf{S}}_{n-1,c,k,l}]$ where n is number of most probable ACSS sequences to be predicted. Actually, we decode $\boldsymbol{\mathcal{C}}_{c,k,l}(\hat{\mathbf{T}}_{c,k,l})$ n times with a special heuristic algorithm, which is described in section III-D, and store the resulting ACSS data in $\hat{\mathbf{S}}_{n-1,c,k,l}$. For example, $\hat{\mathbf{S}}_{0,10,3,3,5}$ represents ACSS of pixel coordinate (3, 3) at future time-step 5 belonging to the most probable future ACSS sequences, which is predicted at time-step 10. We define the elements of output tensor $\mathbb{Y}$ as $\mathbf{Y}_i = [\hat{\mathbf{S}}_{i,:,:,:,0}, \ldots, \hat{\mathbf{S}}_{i,:,:,:,j}]$, where $\mathbf{Y}_0$ is the sequence representing the most probable future ACSS sequence.

Note that we only need to decode a set of first-order Markov chains whose transition probabilities are learned by $\mathcal{N}$ in training time, which makes the proposed method a very efficient approximation of HMMs.

C. THE NEURAL NETWORK MODEL

The first part of our model $\mathcal{N}$ (Fig. 3) contains 3 layers of 2D CNNs with 128 filters of size 5×5. We define a 1-step stride along each dimension for these layers. After each convolutional layer, we add a 2D max pooling layer with a pool size of [2, 2], and a stride of [2, 2]. The output of the last max-pooling layer is fed to a Fully Connected Neural Network (FCNN) of two layers with dimensions 2048 and 96 units, respectively. The resulting visual feature vector of size 96 and the actional feature vector of size 8 ($\mathbf{a}_i$) is concatenated and fed to an LSTM module.

We employ an LSTM module with 4 layers, each of which contains 512 hidden nodes as a sequence prediction model. Input to LSTM is a vector whose size is the sum of the number of visual features and actional features ($96+8$). The output is a matrix $\mathbf{O_i}$, whose rows correspond to each future time-step of the predicted sequences. The number of future time-steps can be freely determined within the memory limitations of available GPUs. Each column of $\mathbf{O_i}$ represents the state transition probabilities of ACSSs and action predictions. The last $|A|$ columns of $\mathbf{O_i}$ correspond to $\hat{\mathbf{a}}_j$ which is a probability distribution representing what actions are expected in time-step t_j for the whole frame.

The output of the LSTM module $\mathbf{O_i}$ contains the state transition probabilities of ACSSs in an encoded form. While action prediction for a future frame is already estimated by the LSTM module, state transition predictions should also be decoded. We feed encoded transition probabilities to an FCNN of two layers with dimensions 2048 and 96 units, respectively. Then, the output of the FCNN is fed to a 3-layered 2D Deconvolutional Neural Network (DecNN) whose outputs are the state transition probabilities of ACSS states $\hat{\mathbf{T}}$. We set all parameters the same as those used in the CNN module. We used a combination of *tanh*, *relu* and *leaky − relu* as activation functions in all the layers.

Loss function. The loss function consists of two components: action loss $\mathcal{L}_a$ and state transition loss $\mathcal{L}_t$. $\mathcal{L}_a$ is used to measure the accuracy of the actional predictions for a frame by comparing the predicted action data $\hat{\mathbf{a}}_j$ against the one-hot Ground Truth (GT) vector $\mathbf{a}_j$. We use

$$\mathcal{L}_a(\mathbf{a}, \hat{\mathbf{a}}) = SCE(\mathbf{a}, \hat{\mathbf{a}}) \tag{1}$$

to measure the difference between $\hat{\mathbf{a}}_j$ and $\mathbf{a}_j$, where SCE is the softmax cross entropy which is defined as

$$SCE(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=0}^{n} \mathbf{y}_i \log(\mathrm{softmax}(\hat{\mathbf{y}}_i)), \tag{2}$$

where $n = \dim \mathbf{y}$.

The second term of the loss function $\mathcal{L}_t$ computes state transition likelihood error between the GT state transition matrix $\mathbf{t}$ and the predicted state transition matrix $\hat{\mathbf{t}}$. It computes the softmax cross-entropy between predicted and GT probability distributions of state transitions for each pixel coordinate of a predicted frame and calculates average loss for the frame.

$$\mathcal{L}_t(\mathbf{t}, \hat{\mathbf{t}}) = -\frac{1}{whm} \sum_{i=0}^{h} \sum_{j=0}^{w} \sum_{k=0}^{m} SCE(\mathbf{t}_{ijk}, \hat{\mathbf{t}}_{ijk}), \tag{3}$$

where $m = 3|A|$, h is image height and w is image width. The joint loss function is defined as

$$\mathcal{L}_f(\mathbf{a}_j, \mathrm{T}_j, \hat{\mathbf{a}}_j, \hat{\mathrm{T}}_j) = \lambda \mathcal{L}_a(\mathbf{a}_j, \hat{\mathbf{a}}_j) + \mathcal{L}_t(\mathrm{T}_j, \hat{\mathrm{T}}_j), \tag{4}$$

where λ is the weighting parameter for balancing the two loss terms. In practice, the weighting parameter is set as $\lambda = 1$ empirically.

$\mathcal{L}_f$ defines a loss function for a single future time-step t_j. The loss for a whole sequence of future time-steps is defined as

$$\mathcal{L}(\mathbf{a}_c, \mathbf{T}_c, \hat{\mathbf{a}}_c, \hat{\mathbf{T}}_c) = \frac{1}{L} \sum_{j=0}^{L} \mathcal{L}_f(\mathbf{a}_{cj}, \mathbf{T}_{cj}, \hat{\mathbf{a}}_{cj}, \hat{\mathbf{T}}_{cj}), \tag{5}$$

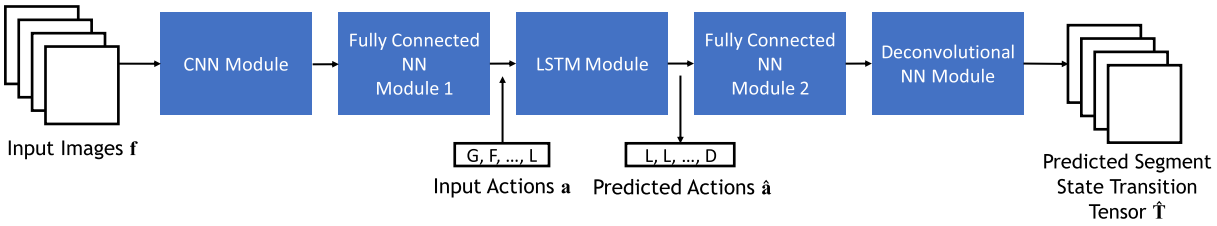


FIGURE 3. Our neural network model $\mathcal{N}$.

where $\mathbf{a}_c$ and $\mathbf{T}_c$ are GT future action and state transition sequences starting from time-step t_c, respectively. $\mathbf{T}_c$ is computed using GT operator actions and RGB frame segmentations. Note that, for all of our formulations, the terms with a cap represent corresponding predictions. Finally, the model optimization function is defined as

$$\mathcal{N}^* = \arg\min_{\mathcal{N}} \mathcal{L}(\mathbf{a}, \mathbf{T}, \mathcal{N}(\mathbf{X})_a, \mathcal{N}(\mathbf{X})_T) \tag{6}$$

Adam optimizer is employed with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and learning rate of $1e-4$ in the optimization process.

D. FUTURE SEQUENCE GENERATION

Since we need to find top n most probable future sequences, it is not possible to use optimal methods such as standard Viterbi algorithm as they would generate only the best future sequence. Instead, we develop a new heuristic method to find the most probable future sequences. Even though it is not guaranteed to produce the optimal future sequences, the final results satisfy our problem requirements, which we validate through extensive experiments on human operators.

In the first step of our heuristic decoding procedure, $\mathcal{N}$ predicts $\hat{\mathbf{T}}_c$ by evaluating the input sequence of operator actions and the images at time-step t_c (Fig. 2). Then, we start iterating through the Markov chains $\boldsymbol{\mathcal{C}}_c$. We create a set of likelihood images $I_{BG-F}, \ldots, I_{FG-D}$ of size $w \times h$ for each of the ACSS categories by using the transition probabilities at each iteration. For example, the transition probabilities of the states representing $FG-F$ in Markov chains $\boldsymbol{\mathcal{C}}_{c,:,:,0}$ are used to create image I_{FG-F} at the first iteration. Note that transition probabilities are updated at each iteration, which makes it possible to generate different images at each iteration. The pixel values of the images are obtained from corresponding pixel coordinates of the transition probabilities $I_{j,:}(k,l) = \boldsymbol{\mathcal{C}}_{c,k,l,j}$. We find the images containing a foreground region by checking $\hat{\mathbf{a}}_c$. Then, we run a blob detector on each of these images to locate the foreground regions. We find centroids of each of the foreground regions and get their transition distributions. The state transition distribution of a centroid indicates which types of actions are likely to affect that pixel coordinate in the next time-step. If the transition probability for a next state is greater than ε, which is determined as 0.1 empirically, we iterate to that state for all the Markov chains $\boldsymbol{\mathcal{C}}_{c,:,:,j}$. We record the action type and the likelihood image before we iterate to the next state. If there are multiple probable states at an iteration, we fork the process for each of the possible states. We perform the same procedure for the subsequent iterations. In the end, we come up with a set of predicted future sequences containing ACSS images. We compute sequence probabilities by multiplying all ACSS state transition probabilities in a given sequence. Note that our model is a special case of the Markov chain, and we should expect that the joint probabilities of all possible future sequences should sum up to 1.0. We sort the produced future sequences and get the top 5 most probable ones in terms of sequence probabilities to form the actional and visual suggestion list $\hat{\mathbf{S}}_{n,c,k,l,j}$. Note that our output is a set of ACSS sequences, but it is trivial to convert them to VSSI sequences, which are shown to the operators as visual suggestions.

FIGURE 4. Snapshots of our real robot in an experimental teleoperation environment.

By employing such a stochastic future sequence prediction approach, our system can handle uncertainties in future sequences. SATS lists all the alternatives in such cases, and the operators choose an appropriate suggestion to resolve any uncertainty in future sequences. That is one of the main advantages of our method.

E. IMPLEMENTATION DETAILS

We used GT segmentations generated by the synthetic environment as training data. Before training the model, all the input data are normalized between $[-1, 1]$ to facilitate the training process. The proposed method is implemented using TensorFlow library with Python language. Training of the model takes $30-35$ hours on a GTX 1080 Ti GPU. SATS can generate future sequences in less than a second, thanks to its efficient prediction module running on GPU, which makes it very practical in real-world applications. Although we defined 24 different ACSS labels for hidden states of $\mathcal{C}$, we use only 10 labels in the implementation to facilitate the training process. Actually, we omit all ACSS labels belonging to the robotic arm and background $BG-F, BG-L, \ldots, RA-D$ as we are not interested in what actions affect the

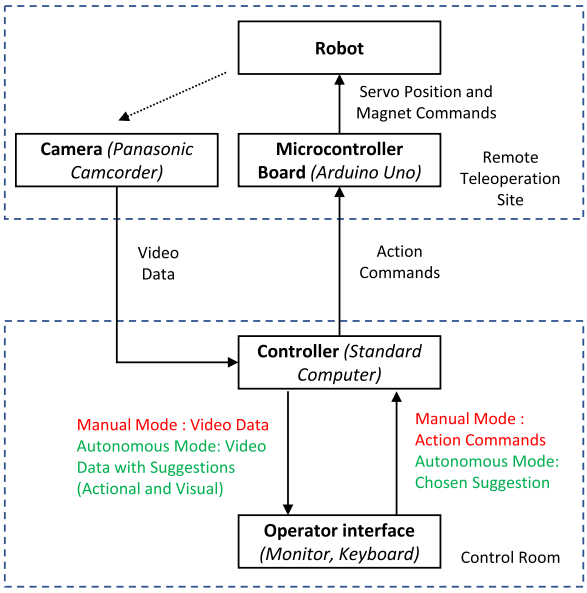


FIGURE 5. Control architecture of our model.

robotic arm and background regions in practice. The multiple future sequence prediction process can be done using only ACSS information about the foreground regions in our application, which is why we omit ACSS labels related to the robotic arm. We used only one ACSS label for each of the robotic arm and background regions and 8 for the foreground regions, which sum up to 10 labels.

IV. EXPERIMENTS

A. TELEOPERATION ENVIRONMENT

1) REAL SCENE

We build a miniature crane robot to simulate a typical teleoperation system (Fig. 4, Fig. 5). The robot consists of an aluminum frame, several servo motors, a power supply, and an electromagnet, which are controlled by an Arduino Uno microcontroller board. We assemble servo motors in the directions of the X axis, Y axis, and Z axis, so it has three degrees of freedom. We use plastic blocks and cylinders to represent the foreground objects. We placed magnetic metal pieces on their top to grip the objects using the electromagnetic gripper. We also placed a camera in front of the scene to provide visual feedback to the operator (Fig. 6-a). The system is connected to a remote computer so that the operator can send various commands (actions) to control the robot. In order to perform an action sequence, the operator grips an object using the electromagnetic gripper of the robot, moves it to the target location, and finally releases it. In order to make our environment more realistic, other than covering the backside of the scene with cardboard, we did not try to eliminate shadows or other background entities such as cables.

2) SYNTHETIC SCENE

The real robot is very useful for evaluating our final system with real operators. However, for convenient evaluation of initial ideas and for producing training data for the neural model, we also implemented a synthetic 3D teleoperation scene using the Unity3D engine (Fig. 6-b). The virtual environment is modeled roughly similar to its real counterpart, and the virtual robot provides almost the same functionality as the real robot. Different from the real robot, the virtual system has an additional servo motor to rotate the electromagnetic gripper around the Y axis; therefore, it has four degrees of freedom. The operator can move and manipulate scene objects using a keyboard. Our virtual robot automatically generates the raw RGB frames (Fig. 6-b) and their GT segmentations (Fig. 6-c). We record this data along with the corresponding operator actions during the operation of the virtual robot.

Although it might be preferable to use a more sophisticated real-world robot mechanism with a more advanced control scheme in such a study, we believe that our system is adequately sophisticated to prove the validity of our novel ideas. Let us consider transportation and object stacking-based teleoperation tasks such as removing nuclear debris from a nuclear disaster site, constructing basic settlements on a distant planet, or gathering raw material from an asteroid. Such operations include phases of gripping an object, carrying it to a target point by following a specific track, and releasing it onto the destination, each of which can be performed by using the functionalities that our system provides. Besides, other properties of the robotic mechanism such as velocity or acceleration are relatively less important in such systems because it typically moves at a constant speed through transportation.

B. DATASETS

We created 4 different datasets: Real, Synthetic General Missions (GM), Synthetic Wall Builder (WB) and Synthetic Pyramid Builder (PB) using the real and synthetic teleoperation environments for evaluating the performance of the proposed system as well as demonstrating its usefulness on practical applications. We have 6 types of action sequences in the Real dataset and 22 types of action sequences in the GM dataset. Fig. 7 shows the projections of the moving object paths of these sequence types on the $X - Z$ plane. Each action sequence type is represented by a different colored arrowhead. The sequences start from the source point S and move to one of the target points (T). We used two types of backgrounds a and b having 4 and 6 types of different action sequence types, respectively. For the Real dataset, we only generated samples of patterns for only backgrounds of type b. On the other hand, GM contains samples of both background types. Annotations placed next to the arrowheads in Fig. 7 indicate in which background that action sequence is performed. When performing an action sequence type b_i, we first perform the sequences with no rotation, which produces 6 instances of different action sequence types. Then, we resample the sequences after applying a rotation to the foreground object with angles of -90 and $+90$, which is performed right after the grip action. In this way, we augment the

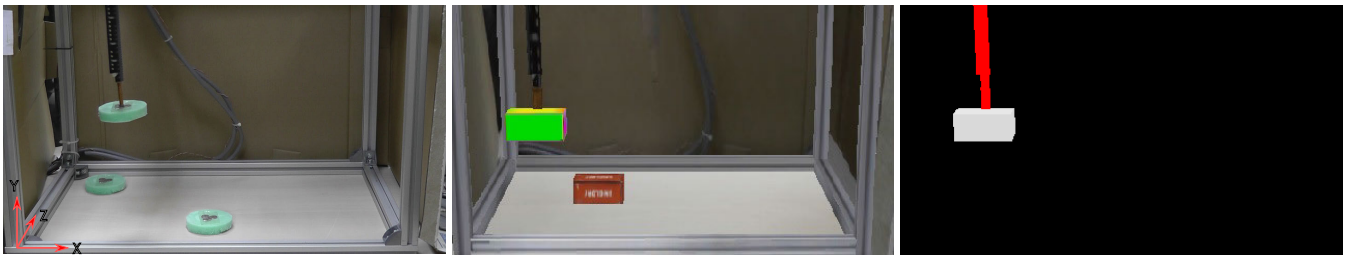


FIGURE 6. A set of samples from our datasets. (a) An image from the real scene, which is taken from our test dataset, (b) an image taken from our synthetic training dataset, and (c) GT segmentation image of (b). We train SATS using the synthetic training dataset and test it using both synthetic and real test datasets.

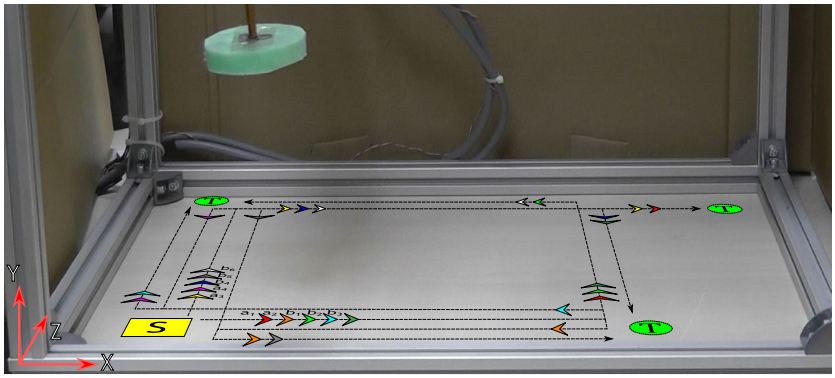


FIGURE 7. $X-Z$ plane projections of action sequence patterns of the teleoperation tasks we used in our scenario. The operator picks objects from a source point S, then carries it to a target point T. We specifically choose these patterns containing many junction points to generate data having splitting multiple future sequences.

number of type b action sequences from 6 to $6(no\ rotation)+6(+90\ degree\ rotation)+6(-90\ degree\ rotation)=18$. Since GM also has the action sequence types of a, the total number of action sequence types of GM is 22. Note that the Real dataset contains no rotations.

In order to create large training datasets using our virtual robotic environments, we designed an autopilot sequence generation tool that takes an action sequence type and produces many sequence instances of the same type with varying time-step sampling rates and sequence lengths. SATS uses a standard time-step duration of $\tau = 1$ seconds system-wide. In order to realistically model the latency differences between the operators, we add uniformly distributed $\pm 10\%$ noise to τ for each time step. We group consecutive steps with the same primitive action type to form a *subsequence*, which is represented as a line in Fig. 7. Our autopilot tool models a subsequence with parameters of action type and Normally distributed subsequence length $\mathcal{N}(\mu, \sigma)$. We set $\mu = 10$ and $\sigma = 2$ for both Real and GM datasets. A sequence is a combination of several subsequences. The sequences in Real and GM datasets contain an average of 3-4 subsequences. A generated sequence starts from the source location S, where the scene objects are waiting to be carried. After the sequence is completed, the object being carried is released on a target point T and the robot returns to S to perform the next sequence.

We generated 50 instances for each of the 6 sequence types in the Real dataset. Therefore, the total number of sequence instances is 300. We have about 3 subsequences for each sequence instance on average, and the average number of actions in a subsequence is roughly 11, making the total number of primitive actions, i.e., time-steps, in Real dataset 9900. In GM, there are 100 instances for each sequence type that makes the total number of sequence instances 2200. Since the above calculation procedure also applies to GM, the total number of primitive actions in GM is 75000.

For the WB dataset, we designed another synthetic 3D environment in which an operator manipulates a robot to build a wall by stacking bricks one by one (Fig. 16). We used a different configuration of the autopilot tool to generate actional and visual data using this 3D environment. Similar to the GM dataset generation procedure, the subsequences produced for this dataset use the same noisy τ and subsequence length models. In the resulting dataset, we have several hundreds of wall-building sequences. We applied a similar procedure for creating the PB dataset.

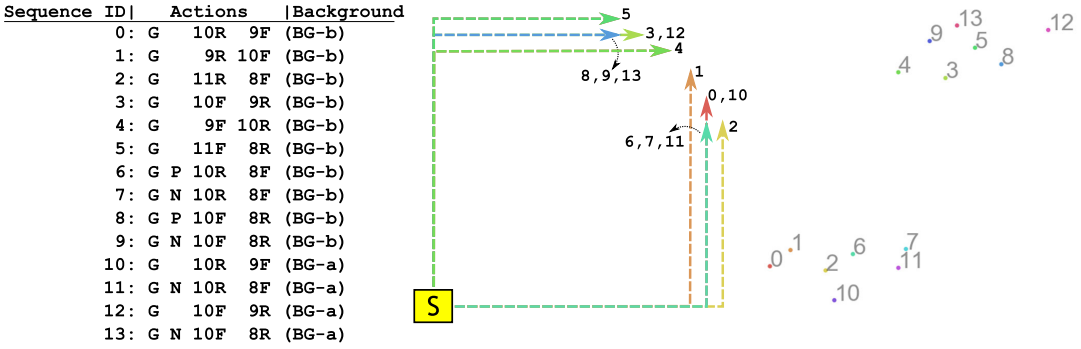


FIGURE 8. Visualization of hidden state vectors sampled by various teleoperation tasks using t-SNE technique. **Left.** List of performed action sequences. BG-a and BG-b denote slightly different backgrounds. **Middle.** Visual representation of the action sequences. These are only rough representations of the input sequences. Actual data have a relatively huge variation because of the random noise added. **Right.** 2D t-SNE representations of the hidden state vectors.

C. VISUALIZATION OF HIDDEN LSTM STATES

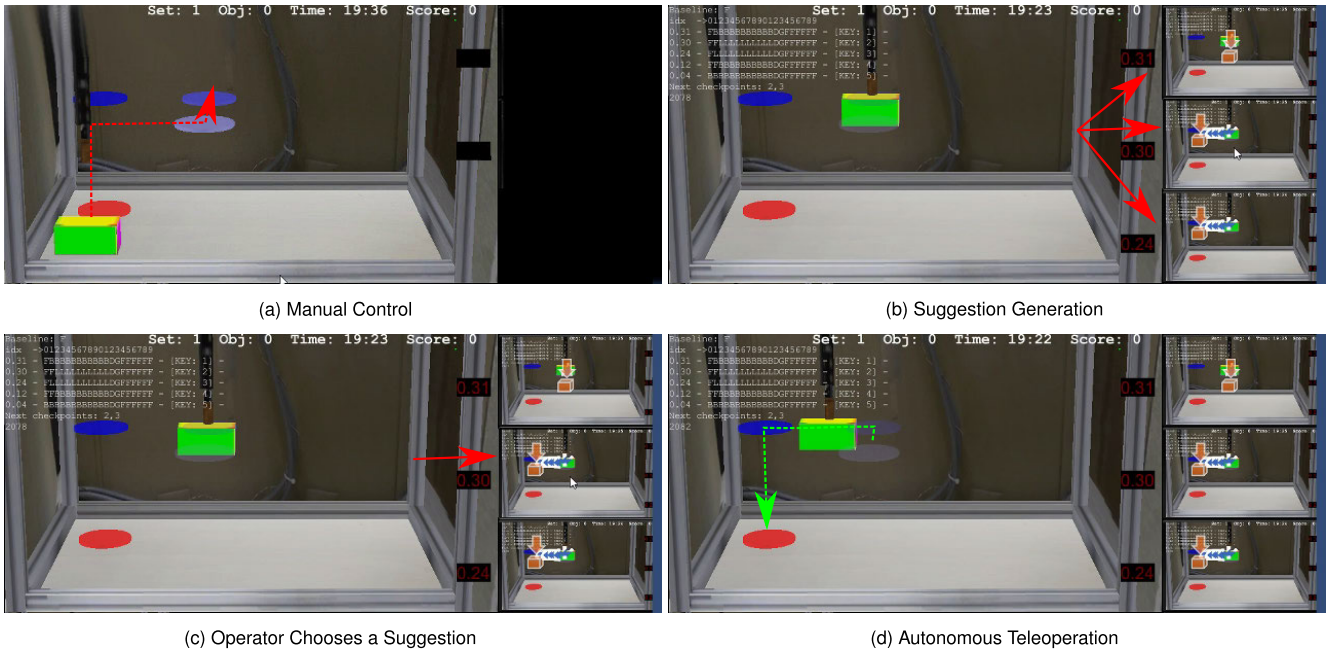
After training our network model $\mathcal{N}$, we visualized the hidden state vector of the last LSTM layer to demonstrate that our model captures the meaningful features of the actional and visual input sequences. To do so, we defined a set of action sequence types, which are listed in Fig. 8-Left. The $X-Z$ plane projections of the same sequences are displayed in Fig. 8-Center. We performed each of the action sequence types to create a dataset for visualization. For example, the operator issues 1 G, 10 R, and 9 F action commands sequentially to perform an instance of the sequence 0. As we did before, each sequence is produced with automatically added noise. We repeat sampling each action sequence type 5 times to generate a relatively well-distributed dataset. We feed the collected action and image data to the neural model and record the hidden state vectors of size 512 of LSTM after each action sequence is performed. Then, we compute the centroids of the vectors belonging to the same action sequence types. We employed t-Distributed Stochastic Neighbor Embedding (t-SNE) technique [42] to reduce the dimension of the centroids from 512 to 2. Fig. 8-Right visualizes the resulting 2D centroids for each sequence type. It can be seen that similar input sequences are close to each other and distant from different types of sequences (e.g., $0-1-2$ and $3-4-5$).

In order to investigate the effects of different backgrounds on the hidden state vectors, we compared inputs having the same action sequences but slightly different backgrounds, i.e., performing the same action sequence in a partially modified background. For instance, sequences 0 and 10 have the same type of actional data but slightly different visual data. As a result of this, centroid 10 is far from centroids 0, 1, and 2 compared to centroids 3, 4, and 5. However, it is a little bit distant from the $(0-1-2)$ cluster because of the visual differences in the background. The same relation also exists between centroids 3 and 12.

The approach we applied in this experiment is closely related to popular word representation techniques, such as word2vec [43], which is widely used in the Natural Language Processing context. In word2vec, words are converted to vectors in a way that words that share common contexts are also close to each other in the vector space. Similarly, our method takes actional and visual sequences and produces a vector space with each sequence being assigned a corresponding vector in this space. With this experiment, we can argue that SATS captures meaningful input sequence information to make similarity measurements between them. We can also argue that our system can behave sensibly even for cases where the input sequence was not seen before.

D. SUGGESTION GENERATION AND ASSISTIVE TELEOPERATION

SATS acts as an interface between the teleoperation environment and the operator. It continuously collects data from the teleoperation environment and generates suggestions to help the operators while they perform operations. SATS receives actional and visual input data at each time-step (Fig. 9-a) and predicts a set of possible future sequences (Fig. 9-b). The operator side application retrieves the prediction sequences and shows them on the screen by inserting various icons onto the centroids of the predicted foreground object segments. These suggestions are listed as images on the right of the screen ordered by their sequence probabilities (Fig. 9-b). The sequence probabilities are also displayed on the left of each suggestion. The color of the probability text is proportional to its probability value ranging from black to red (black for 0 and red for 1). After the operators evaluate the suggested actions, they can choose a suggestion (Fig. 9-c) to make SATS perform suggested actions autonomously (Fig. 9-d). SATS keeps collecting actional and visual data while carrying out the issued task autonomously. After it completes the task at hand, it instantly generates a new set of suggestions for the next task based on the data collected during the autonomous control. Again, the operator checks the new suggestions and chooses a suggestion to activate SATS. If no proper suggestion is listed on the screen, the operator can simply continue to control the robot manually until SATS generates a desired suggestion.



(a) Manual Control

(b) Suggestion Generation

(c) Operator Chooses a Suggestion

(d) Autonomous Teleoperation

FIGURE 9. A step-by-step usage demonstration of the proposed system. (a) The operator uses the robot manually for a while. (b) SATS generates a set of suggestions based on the data collected during manual control. (c) The operator chooses a suggestion to activate SATS. (d) SATS completes the remaining task autonomously.

TABLE 1. Command (Action) automation fates (CAR) for the available datasets. The last column gives average CAR for all the datasets. Other columns represent CARs for each individual action type. A CAR value of 1.0 is equal to 100% autonomy.

Dataset	Grip	Forward	Backward	Left	Right	Drop	Positive Rot.	Negative Rot.	Overall
General Missions	0.90/0.05 (μ/σ)	0.95/0.03	0.95/0.03	0.95/0.02	0.94/0.04	0.95/0.03	0.88/0.05	0.89/0.04	0.93/0.04
Wall Builder	0.91/0.04	0.96/0.02	0.97/0.01	0.97/0.01	0.96/0.02	0.96/0.02	-	-	0.96/0.02
Pyramid Builder	0.90/0.05	0.96/0.02	0.96/0.03	0.97/0.02	0.96/0.02	0.95/0.03	-	-	0.96/0.03
Real Robot	0.88/0.04	0.92/0.03	0.90/0.04	0.91/0.04	0.92/0.03	0.91/0.04	-	-	0.91/0.04

E. PERFORMANCE EVALUATION

SATS is designed to assist teleoperation operators in performing their tasks more effectively. In order to evaluate SATS in this context, we used 5 volunteer operators to test it on both synthetic and real data. We believe that, at least for the first stage, the pool of 5 volunteers is sufficient to assess the performance of SATS. There are two reasons for this view. Firstly, we think that SATS performance can be measured using quantitative metrics that are independent of the size of the volunteer pool. As we discussed earlier, our study is mainly focused on automating the teleoperation process as much as possible and estimating additional information about the ongoing teleoperation for the operators' use. This is an objective whose success rate can be measured, to a certain degree, by employing quantitative metrics such as command automation rate, dice similarity/F-score, etc. We, therefore, just aimed to maximize and measure those metrics for now. Secondly, we observed that many similar studies in the literature use five subjects, such as [14], [44] that show us our volunteer pool is adequately enough to validate the capabilities of SATS at its early phases. As we primarily aimed to show the potential usefulness of SATS as an AI-powered assistive system in teleoperation, we did not focus on analyzing human subjects deeply. We left issues like questioning the volunteers about their experiences with SATS for future work for this reason.

The SATS instance that is used in the experiments is trained on only 80% of the synthetic data. Each operator is trained to familiarize themselves with the teleoperation environment and robot manipulation by showing them the robot controls, interface details, and SATS commands. They are allowed to spend some time with the overall environment until they are confident with their teleoperation skills.

1) COMMAND AUTOMATION RATE

In the first experiment, we assessed the action prediction capabilities and automation efficiency of SATS. The operators are asked to complete as many teleoperation tasks as they can in a 20-minute period. They are also required to use SATS whenever possible (when there is a usable suggestion) while performing tasks. We record manually issued commands (i.e., actions) as well as autonomously issued commands

during the whole teleoperation. Command Automation Rate (CAR) is defined as the ratio of the autonomous command count to the total number of the issued commands in a teleoperation mission. We computed the mean and the standard deviation of CAR values for each of the datasets which are shown in the last column of Table 1. The CAR results show that our system can achieve more than 90% automation rates regardless of the dataset, which means the proposed method can greatly reduce the number of manual commands required to complete a teleoperation mission in both real and synthetic environments. Table 1 also gives CAR results for each of the individual action types, which shows that SATS performs favorably regardless of action types. We also see that some of the action primitives, such as the rotations, may need more operator attention, which might be due to a lack of training data for these types of actions.

2) COMPARISION WITH A BASELINE HIDDEN MARKOV MODEL

In this experiment, we compared the action prediction capabilities of the proposed method with a baseline Hidden Markov Model (HMM). This model observes the sequence of the action commands issued by the operator and estimates future actions. In other words, it observes actions starting from time-step t_{i-c} to t_i and estimates actions from t_i to t_{i+L} at each time-step, where c and L are defined as the length of the observation and the prediction sequences, respectively. While the issued actions represent observations, the estimated actions represent hidden states. This implies that both the observation types and hidden state types are the same for this model: G, F, B, L, R, D, P, N. We computed the starting, the transition, and the emission probabilities of the HMM by processing our original simulative *General Missions* dataset, which was used to train SATS previously. Once we computed all the parameters of the HMM, we tested it on our simulative teleoperation environment. We first generate a sequence of actions using the virtual robot. Then, we feed the generated action sequence to the HMM as an observation sequence. Next, we decode the HMM and find the best sequence of hidden states explaining the observation sequence. The output hidden state sequence is the expected action sequence for future time-steps. We predicted 61.450 sequences using the HMM and compared the estimated action sequences with the ground truth data. We observed that the HMM could achieve a 36.87% accuracy rate on our simulative dataset. This is a poor performance compared to the SATS. Please note that SATS achieved approximately 95% accuracy in the same dataset as reported in the last column of Table 1.

3) INCREASING OPERATOR THROUGHPUT WITH MULTIPLE ROBOTS

Our first experiment showed that more than 90% of the issued actions can be automated, which means the operator will stay idle for a long time. SATS would utilize the operator time more efficiently if we can benefit from the available idle time of the operators. We introduce a new operator efficiency metric, the Operator Throughput (OT), which is defined as the number of tasks completed in a 20-minute period. As a baseline OT measurement, we have created a virtual teleoperation environment and asked the operators to complete a mission manually without using SATS. The measured OT values for this experiment are reported in the first row of Table 2. As expected, the OT values did not increase much (the second row of Table 2) when the operator is asked to complete the same mission using SATS because the operator stays idle during the available free time and the overall time for a task would be about the same. In order to increase the throughput of the operators, the available free operator time can be utilized by letting the operator control multiple robots at the same time. For this purpose, the operator is asked to complete similar teleoperation missions simultaneously with the help of SATS using multiple robots (3 virtual robots located in independent environments). Fig. 10 explains this experiment by providing snapshots from the teleoperation scenes. Please also see the provided supplementary video material of the same experiment. In the beginning, the operator uses robot 1 manually until SATS produces suggestions for automation for this robot. The operator chooses the desired suggestion, and while SATS executes the suggestion, the operator can switch to the second environment. After the operator performs teleoperation in a 20-minute period using multiple robots, we calculate total OT for all 3 missions, which is reported at the last row of Table 2. Fig. 10 shows that there is always a task waiting to be handled by the operator (snapshots marked with "Needs Attention"), and the operator is always active for each step, which results in a throughput increase of more than 90% compared to manual control.

TABLE 2. Operator throughput of different teleoperation configurations.

	Avg. Throughput	Boost Rate
Manual Control	32.7/0.95 (μ/σ)	-
Single (SATS Full)	33.0/1.25	1.04%
Single (SATS without Vis. Cue)	25.2/2.98	-22.94%
Multi-Robot (SATS Full)	**62.8/1.89**	**92.05%**

While the increase in OT is very promising, we like to stress the importance of the Human-Computer Interface (HCI) and usability design of the overall system. The current system interface design is based on the assumption of the importance of the visual information, which is verified by running an experiment in which the operator uses SATS to complete a mission without any visual cues about the predicted suggestions. Instead of visual cues, the operator only sees a sequence of text showing the prediction action suggestions. The experimental results showed that OT decreases dramatically compared to manual control (the third row of Table 2) when the operator cannot see the future sequences visually.

We also observed that the most time-consuming part of operating multiple robots is the operator perception time of the SATS suggestions, which includes operator context

FIGURE 10. Visualization of simultaneous multi-robot control using the proposed system. Each column shows snapshots from a different environment. Each row demonstrates a step while the operator uses SATS (Refer to supplementary material for more results).

switch time between robots. The current HCI design of the system is developed just to demonstrate the idea of multiple robot control without much consideration for the operator's mental loads and interface usability. If the optimal HCI and usability parameters of the system are selected, it might be possible to gain better OT numbers and to integrate more robots into the system to increase the system efficiency.

The performance of the operators is also affected by other factors. If the mental loads of the operators are too low, which is called underload, their attention level will decrease considerably [45], which results in an OT decline. When the operators use multiple robots, SATS continuously assigns a new task to them to keep the mental loads of the operators at an acceptable level, which prevents a possible OT decrease.

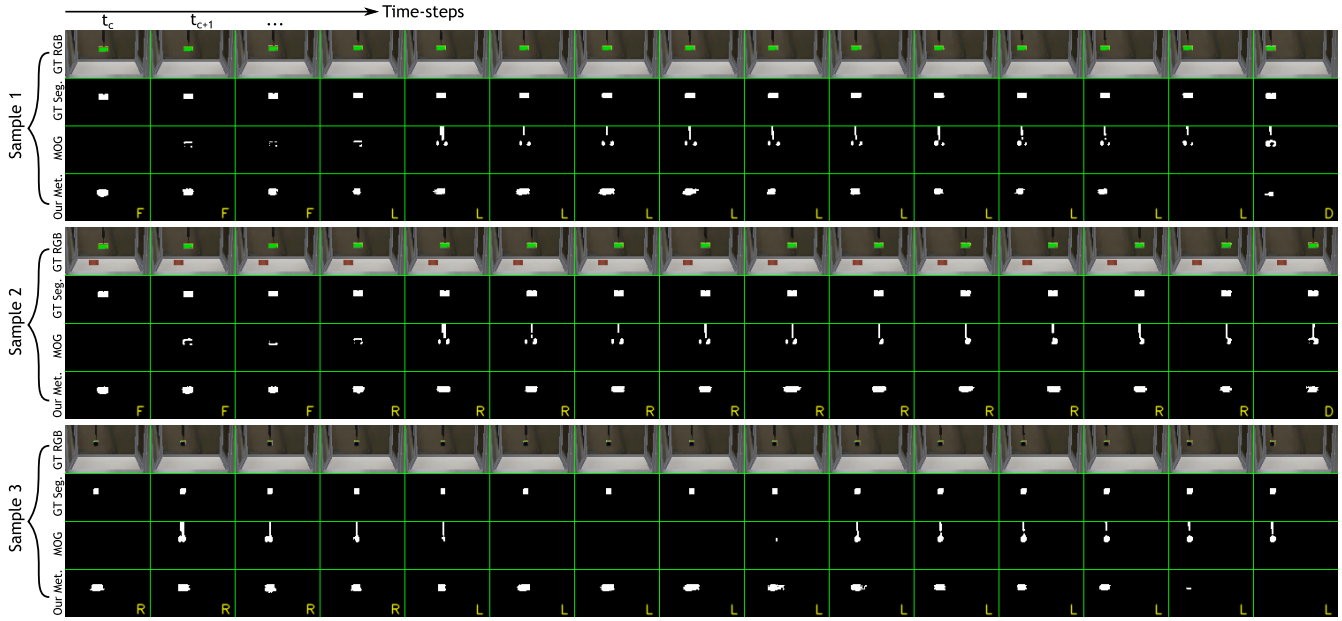


FIGURE 11. 3 samples of future sequence segmentation. **First Rows.** Future GT RGB images, **Second Rows.** Future GT foreground segmentation images, **Third Rows.** Foreground segmentation results generated by baseline MOG background subtractor. **Last Rows.** Predicted future sequence segmentations generated by our method.

TABLE 3. Quantitative segmentation results on synthetic and real datasets. Note that, the SATS instance used in the real experiment is trained on synthetic data and tested on real data.

	Synthetic			Real		
	Precision (μ/σ)	Recall	Dice Similarity	Precision (μ/σ)	Recall	Dice Similarity
MOG	0.46/0.26	0.47/0.29	0.42/0.20	**0.36/0.31**	0.60/0.26	0.37/0.24
Our Method	**0.66/0.25**	**0.84/0.28**	**0.71/0.29**	0.33/0.07	**0.71/0.15**	**0.45/0.08**

4) EVALUATING SEGMENTATION QUALITY

As we mentioned in previous sections, the visual information presented to the operator is essential for better OT values. In this experiment, we measured the performance of the visual prediction (i.e., semantic segmentation) capability of the proposed method. The procedure we used in this experiment can be described as follows. SATS generates a set of future sequences while the operator performs a teleoperation task in a synthetic or real environment. The operator checks the suggestion list and chooses an applicable suggestion to activate SATS. With this experiment, the semantic segmentation of the chosen sequence is compared with the corresponding GT segmentation sequence.

Since the semantic segmentation sequences are probability images indicating how likely their pixels belong to the foreground object, we binarize them with a threshold value of 0.05 to eliminate pixels having low probability values. We run a connected component analysis algorithm to get the foreground segment of the current frame. The resulting segments are compared with corresponding GT segments. We calculate the average accuracy, recall, and dice similarity score of the generated segments and give them in Table 3 for both synthetic and real datasets.

Since there are no well-known actional and visual future sequence prediction techniques available to our knowledge, we compare our segmentation method with a baseline Mixture Of Gaussian (MOG) based background subtraction method introduced by [46]. However, MOG is not designed to generate any future information like our method. In order to make MOG a super advantaged baseline, we directly feed future GT RGB sequences to MOG and get the output foreground segmentation sequences which are used as baseline results. The first row of Table 3 shows the results achieved by MOG. Although providing GT future frames gives MOG a significant advantage over the proposed method, our method outperforms MOG in most of the metrics on synthetic and real datasets.

The qualitative segmentation result can be seen in Fig. 11 and Fig. 12 for synthetic and real datasets, respectively. Each sample future sequence has 4 rows in these figures. The first row in a sample shows future GT RGB sequence frames sampled at each second. The second row presents future GT foreground sequence segmentation frames. The third and the fourth rows show segmentation results generated by MOG and the proposed method, respectively. The GT action type for each time-step is given on the lower right part of the

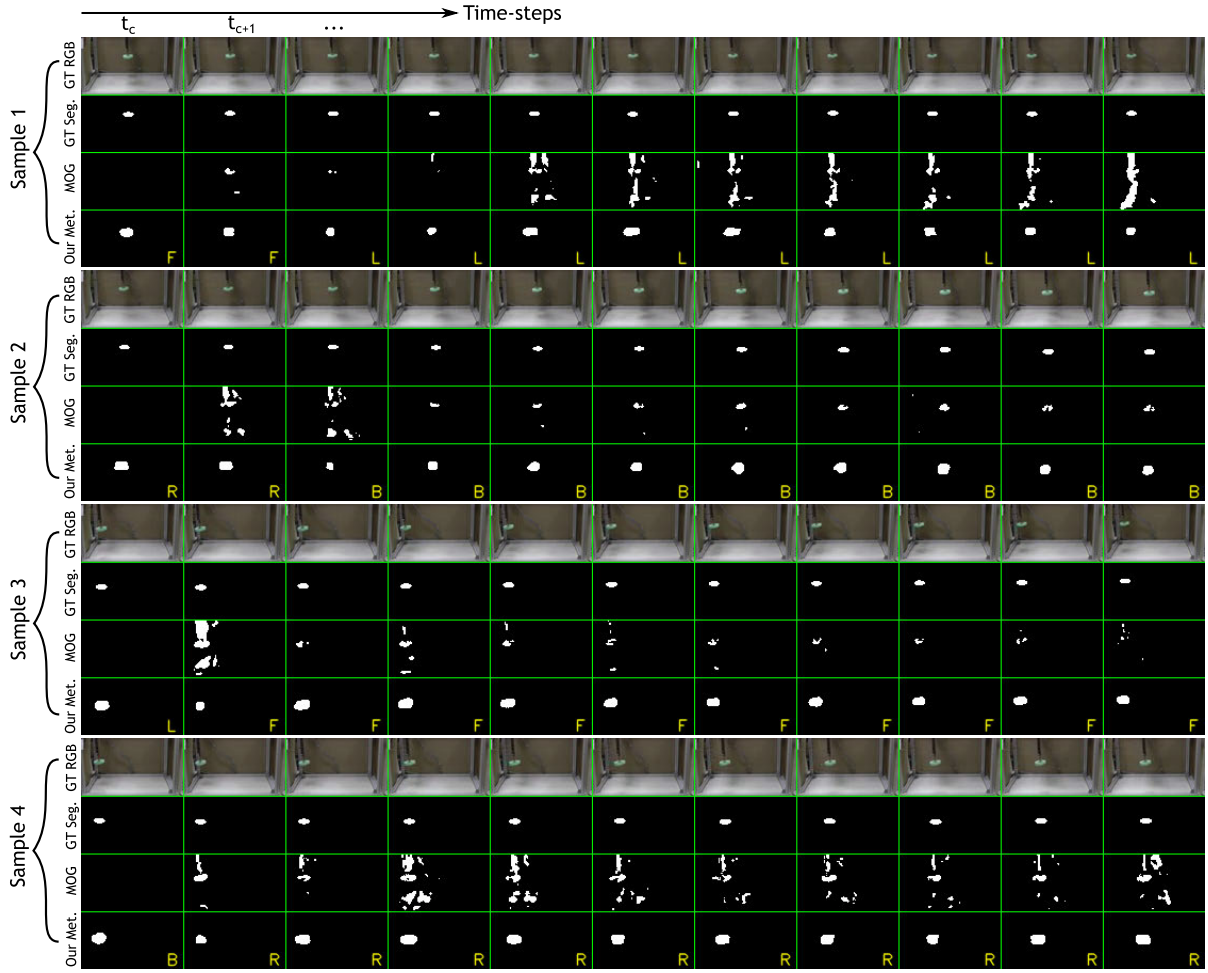


FIGURE 12. 4 sample outputs of future sequence segmentation on the real robot environment. **First Rows.** Future GT RGB images, **Second Rows.** Future GT foreground segmentation images,**Third Rows.** foreground segmentation results generated by baseline MOG background subtractor. **Last Rows.** Predicted future sequence segmentations generated by our method. Note that, the SATS instance used in this experiment is trained on synthetic data and tested on real data. Please see supplementary material for a video demonstration.

segmentation image generated by our method. It can be seen that the proposed method can generate quite compact future sequence segmentations. As expected, segmentations generated by MOG contain mostly the motion regions of the image, which are scattered and inconsistent with GT data. Please note again that while MOG segments only a given future RGB sequence, the proposed method produces segmentations for the complete future sequence.

5) ROBUSTNESS TO UNSEEN ENVIRONMENTS AND VIEWPORTS

The SATS instance employed in experiments on the real dataset has been trained using only synthetic data. Considering the relatively good results that are given in Fig. 12 and Table 3, we can safely conclude that SATS is flexible enough to carry out teleoperation tasks autonomously in unseen/slightly different environments. There are two main reasons for this conclusion. Firstly, there are some moving objects such as parts of the robotic arm and cables in the background in the real test data. There are also shadows in the real data that move as the robotic mechanism moves. None of those visual factors presents in the training dataset. Secondly, the camera parameters in the test environment are not identical to the camera parameters in the training environment. Both intrinsic and extrinsic parameters of the cameras are slightly varied.

In order to clearly show the effectiveness of the proposed method in teleoperation tasks performed in unseen test environments, we performed two additional experiments. We created a new dataset containing video sequences with different backgrounds (Fig. 13-top) and have taken from different viewports (Fig. 14-a and b) to diversify the visual data. After completing each mission, we randomly picked a new background out of three available images and set it as the new background. We also updated the camera pose right after changing the background image. Basically, we pick a rotation angle out of a set of three predefined rotation angles [0, 10, 20] and rotate the camera around the scene by the chosen value. So we captured video data from

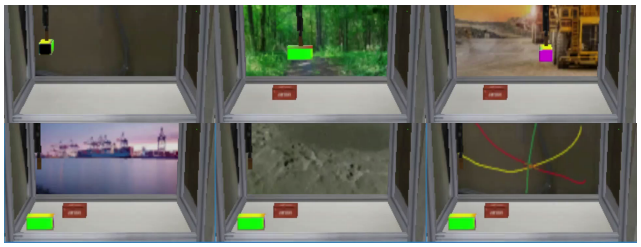


FIGURE 13. Environments used in our experiments. **Top.** Snapshots from environments used in training. **Bottom.** Unseen environments from the test dataset.

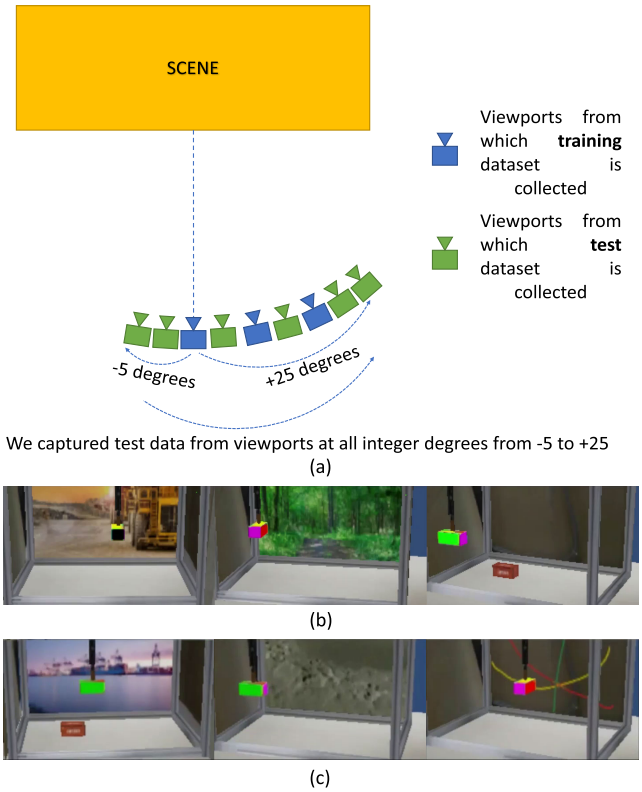


FIGURE 14. Viewports from which our dataset is collected. (a) Positions and orientations of all the viewports used in training and testing. (b) Viewports from which training data is collected. Rotation angles are 0, 10, 20 degrees around the scene, respectively. (c) A bunch of snapshots from unseen viewports from the test dataset. Rotation angles are −5, 16, 25 degrees around the scene, respectively.

three different viewports in total. We retrained our model using this new dataset and performed experiments to measure the performance of our model. In the first experiment, we assessed the robustness of our model to unseen backgrounds. We tested the model with a new test data containing three different unseen background images (Fig. 13-bottom).

In the second experiment, we tested our model with a new set of data collected from various new viewports (Fig. 14-b). More specifically, we rotated the camera around the scene by 1 degree and performed a mission; we repeated this process for 27 iterations from −5 to 25 degrees (viewports used in training are not included). We also changed the background randomly by picking an image from the unseen set of images after each mission. In this way, we collected actional and visual data from 27 different viewports containing unseen backgrounds. We then performed experiments on this data and observed that the proposed method performs relatively well in unseen environments. We provided a bunch of snapshots in Fig. 14-c captured during one of those experiments. (please see the unseenEnvironmentsAndViewports.mp4 video file for details). As a side note, the quantitative results given in Tables 1, 2, 3 and 4 are the average values computed over all the experiments, including this one.

TABLE 4. Waypoint error rates in pixels on synthetic and real datasets.

	Synthetic	Real
	Euclidean Distance (μ/σ)	
MOG	6.10/4.00	6.33/6.86
Our Method	**1.67/0.95**	**3.24/0.80**

6) MEASURING WAYPOINT DETECTION QUALITY

In the final experiment, we measured the waypoint detection capability of our method on future segmentation sequences. A waypoint is defined as the centroid of a foreground segment in a segmentation frame. Accurate prediction of waypoints is important because the icons used in the suggestion images are placed onto the images using the predicted waypoint coordinates. Similar to the previous experiment, we get the predicted future sequences from the suggestions the operators have chosen while performing teleoperation tasks. We obtain future sequence segmentations from the selected suggestion and calculate a waypoint for each of its frames. If no foreground segment is found on a frame, we get the waypoint calculated for the previous frame. Then, we compute the Euclidean distance between the coordinates of the detected waypoint and the corresponding GT waypoint.

We calculate the mean and the standard deviation of these distances and give them in Table 4. Again, we compare our method with the baseline MOG, which is fed with GT RGB future images. The results of our method and MOG are given in pixel errors in the first and the second rows of Table 4, respectively. The first and the second columns of Table 4 show the results of the experiments performed on synthetic and real datasets. It is observed that the proposed method achieves much better results in terms of both the average error and the standard deviation than the baseline MOG method in both datasets.

Predicted waypoints on a number of sample future sequences on synthetic and real datasets are shown in Fig. 15-a and Fig. 15-b, respectively. While each row represents a sample future sequence, each column shows a frame sampled at each second. We used blue markers for GT waypoints, yellow for the waypoints generated by MOG and red

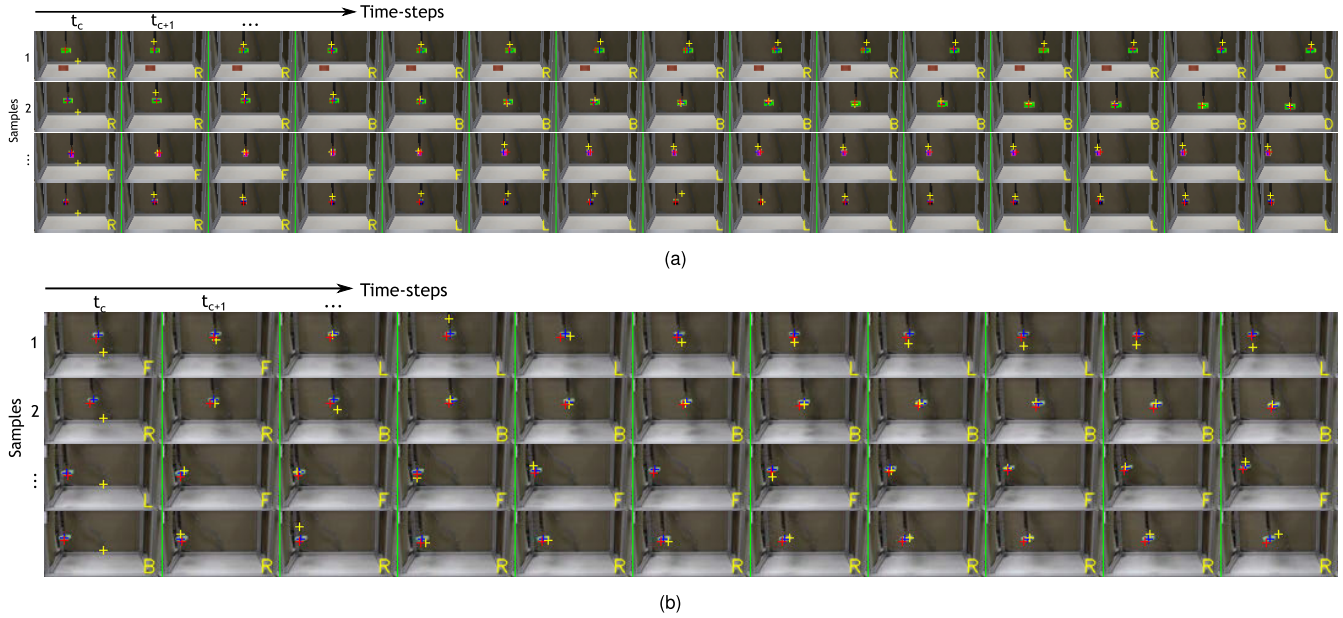


FIGURE 15. GT and predicted waypoints on (a) synthetic and (b) real environments. Blue crosses represent GT waypoints. Yellow and red crosses show waypoints generated by baseline MOG and proposed method, respectively.

for the waypoints generated by the proposed method. The GT actions for each of the future time-step is also given at the lower right part of each frame. Visual analysis of the information presented in these figures agrees with the numerical results reported in 4.

7) SCALABILITY AND RUNTIME PERFORMANCE EVALUATION

We performed an experiment to show that the DoF of our system can be increased with relative ease. When we add rotation capability to our previous 3 DoF synthetic system, we observe that the training time increased from 23 hours to 33 hours. As for the prediction time, it increases by only about 3%. Other costs remain mostly negligible. So, it is possible to increase the DoF of our system at an affordable cost in training time.

We have compared the proposed method with some of the closest alternative techniques in terms of computational cost. [14] is one of the most basic alternative methods which employs a HMM-based model to estimate robot motion. It can generate motion estimations under 1ms. The semi-autonomous teleoperation method proposed in [9] can create predictions at every 60ms. [17] uses a deep neural network to generate visual foresight for planning robot motion, and it can replan the motion at each 200ms. As for the proposed method, the computational cost for a prediction takes about 250ms which makes our method inferior to the alternatives in terms of runtime performance. However, it should be noted that our approach aims at much higher autonomy and provides more complex outputs (future visual estimations, multiple future predictions) than alternative methods. Therefore, it should not be overlooked that this is not a fair comparison.

F. WALL BUILDER APPLICATION

We developed a SATS application to show the effectiveness of the proposed method in practical situations. In this demo application, which we call *Wall Builder*, an operator remotely operates a robot to build a wall by stacking bricks one by one (Fig. 16). We train a SATS model using the WB dataset with the same parameters we defined in Section III. At the test time, operators are asked to build a wall on the scene using provided bricks at the corner of the scene. After the operators carry and place several bricks, SATS starts generating reasonable predictions. The remaining part of the wall-building process can be completed autonomously using the suggestions generated by SATS. Sample snapshots from an assistive wall building process using SATS can be seen in Fig. 16. Please also see the provided supplementary video material of the same experiment. It can be observed that SATS places bricks in a shifted form in subsequent rows in accordance with the training data, which shows that the system produces satisfactory qualitative results. We also reported favorable quantitative results on the WB dataset in the second row of Table 1, which shows very good CAR values achieved by the Wall Builder application.

G. PYRAMID BUILDER APPLICATION

Finally, we performed a more sophisticated experiment using our simulative environment in which the operators can construct pyramids by stacking building blocks one on top of others. In this experiment, there are four types of blocks that can be placed in different parts of a pyramid (top left image in Fig. 17). In order to build a pyramid, the operators need to grip a block using the robot, transfer it to the correct position, and finally release it. We build numerous pyramids

FIGURE 16. Snapshots from the assitive wall builder application. Please see supplementary material for a video demonstration.

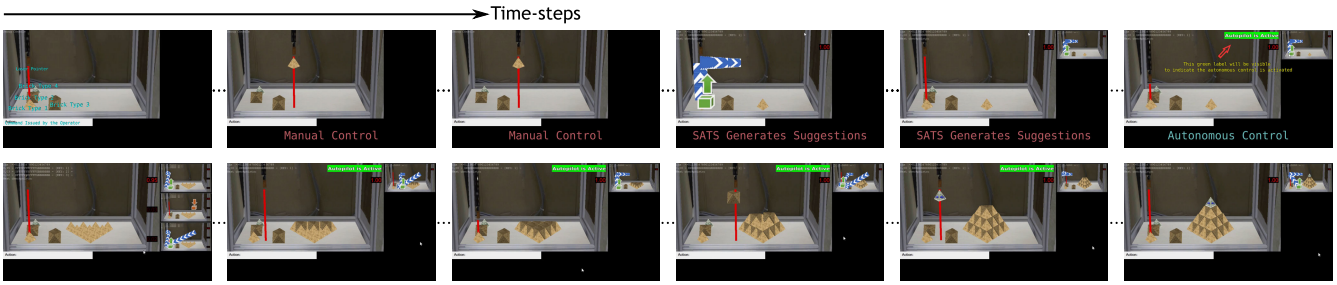


FIGURE 17. Snapshots from the assitive pyramid builder application. Please see supplementary material for a video demonstration.

in our simulative environment and record actional and visual data to create a dataset to train a SATS instance. After we trained a SATS instance using the generated dataset for about 40 hours on a GTX 1080 Ti, we deployed it on the simulative testing environment (Fig. 17). As in previous experiments, the operators control the robot manually for a while at test time to allow SATS to collect some data required for making predictions. Then, SATS starts to generate predictions and offer suggestions to the operators. The operators choose one of the available suggestions to make SATS complete the rest of the task. In this way, most of the pyramid building task is completed autonomously. To be specific, we observed that SATS could autonomously perform **96.14**% of the actions required for building a pyramid. More detailed results are provided in the third row of Table 1. Please see supplementary for a video demonstrating how a pyramid can be built with the help of SATS. We also provide another video to demonstrate how SATS completes the pyramid construction task even if some of the blocks are placed in incorrect positions.

Our system may be seen as an oversimplified design as we abstracted some of the subtasks of classical teleoperation, such as gripping an object or rotating in 3D, etc. However, these experiments show that SATS can handle complex teleoperation tasks in terms of planning and decision-making which are more sophisticated than classical teleoperation tasks. In conclusion, with these experiments, we validated that SATS is an AI-powered assistance system that is capable of helping the operators at higher levels, such as understanding the current phase of the construction and suggesting the next possible actions.

V. CONCLUSION

We presented an end-to-end Stochastic Assistive Teleoperation System (SATS) that continuously monitors the complete teleoperation process and suggests actions to the operators whenever the system detects autonomously applicable actions. SATS does not only predict a sequence of future actions but a set of prediction sequences containing both actions and visual representations of the scene. Employing such an approach allows SATS to achieve several advantages. Firstly, let us imagine a scenario in which an operator carries cargo boxes by following two different paths; path A for sunny weather and path B for rainy weather. Also, suppose A and B are identical in terms of action types, up to a certain step. So without visual processing, it is not possible to decide which path should be suggested to the operator in different weather conditions. Thanks to our design, SATS is capable of generating correct suggestions in such situations. Secondly, the visual output is quite useful for the operators to imagine the expected outcome of the suggested actions, which increases the throughput of the operators. Finally, generating multiple predictions is extremely important when there are multiple possible futures. The prediction subsequences after a future junction point would not be reliable without such an approach. Our novel stochastic prediction mechanism incorporating deep recurrent neural networks and Markov chains makes SATS robust in those kinds of situations. To the best of our knowledge, the proposed system is the first example to employ stochastic action-conditioned video prediction techniques in the autonomous teleoperation domain.

The operators can control multiple robots simultaneously using SATS, increasing the throughput of the operators considerably. In addition, our method generates predicted segmentations and waypoints of the foreground objects for each future sequence frame, which can be employed for further scene analysis tasks.

We validated the proposed system thoroughly on both synthetic and real datasets, which showed that our system

can autonomously perform **93.42**% of the actions required to complete a teleoperation mission. Moreover, SATS boosts the throughput of the operators by **92.05**% as it allows the operators to control up to 3 robots simultaneously. Although this increase in operator throughput is promising, it can be further boosted by employing more sophisticated HCI techniques because we observed that the most time-consuming phase of managing multiple robots is the operator perception time of the generated suggestions. We expect that designing an optimal interface may help achieve lower operator perception times. The experiments on future sequence segmentation and waypoint prediction showed that SATS can achieve more favorable results than the baseline method. The experiments performed with t-SNE methods validated that the neural network model employed in SATS can capture meaningful patterns in the input data.

SATS can be trained with a synthetic dataset collected in an environment roughly similar to the real counterpart and then deployed in the real environment. Considering collecting large amounts of training data from real teleoperation environments is more costly than collecting data from synthetic environments, and most of the teleoperation systems have a simulative operator training environment readily available, this feature could greatly reduce the development cost of autonomous teleoperation systems. This result also makes clear that SATS is flexible enough to carry out teleoperation tasks autonomously in slightly different environments.

Finally, in order to show the effectiveness of our system on practical applications, we created a realistic scenario in which operators build walls and pyramids using simple bricks with the help of SATS. The operators can construct walls and pyramids with automation rates of **96.27**% and **96.14**%, respectively, using the proposed assistive teleoperation system. This result implies that SATS is promising for automating generic industrial applications.

The pyramid building experiment shows us that SATS can be helpful for the operators in different ways. For instance, in the pyramid experiment, certain types of bricks should be placed in a specific order to avoid irreversible mistakes in the construction process. In such scenarios, more attention is required, and it is harder for the operators to recall the correct order of the actions. Fortunately, SATS can suggest the correct actions to operators in such situations with great accuracy, which decreases the mental load of the operators dramatically. Considering that these types of decision-making problems are common in real-life scenarios, it can be said that this would be an important additional feature enhancing the usability of SATS greatly. We plan to focus on validating and improving that kind of features of our method in future work.

REFERENCES

[1] R. C. Goertz, "Mechanical master-slave manipulator," *Nucleonics*, vol. 12, no. 11, Nov. 1954.

[2] W. R. Ferrell and T. B. Sheridan, "Supervisory control of remote manipulation," *IEEE Spectr.*, vol. 4, no. 10, pp. 81–88, Oct. 1967.

[3] R. J. Anderson, "Autonomous, teleoperated, and shared control of robot systems," in *Proc. Int. Conf. Robot. Automat.*, vol. 3, Apr. 1996, pp. 2025–2032.

[4] M. U. Asad, U. Farooq, J. Gu, E. V. Balas, G. Abbas, and M. M. Balas, "Design of a composite state convergence controller for a nonlinear telerobotic system," *Acta Polytechnica Hungarica*, vol. 16, no. 10, pp. 57–112, 2019.

[5] C. Fong, R. Dotson, and A. Bejczy, "Distributed microcomputer control system for advanced teleoperation," in *Proc. Int. Conf. Robot. Automat.*, vol. 3, Apr. 1986, pp. 987–995.

[6] I. Havoutis, A. K. Tanwani, and S. Calinon, "Online incremental learning of manipulation tasks for semi-autonomous teleoperation," Inst. Elect. Electron. Eng., Piscataway, NY, USA, 2016.

[7] A. K. Tanwani and S. Calinon, "A generative model for intention recognition and manipulation assistance in teleoperation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 43–50.

[8] Y. Andrew Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX* (Tracts in Advanced Robotics). Berlin, Germany: Springer, 2006, pp. 363–372.

[9] F. J. Perez-Grau, R. Ragel, F. Caballero, A. Viguria, and A. Ollero, "Semi-autonomous teleoperation of UAVs in search and rescue scenarios," in *Proc. Int. Conf. Unmanned Aircraft Syst.*, Jun. 2017, pp. 1066–1074.

[10] R. F. Fogle, "The use of teleoperators in hostile environment applications," in *Proc. IEEE Int. Conf. Robot. Automat.*, vol. 1, May 1992, pp. 61–66.

[11] Y.-C. Liu and N. Chopra, "Control of semi-autonomous teleoperation system with time delays," *Automatica*, vol. 49, no. 6, pp. 1553–1565, 2013.

[12] A. Kheddar, C. Tzafestas, and P. Coiffet, "The hidden robot concept-high level abstraction teleoperation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, vol. 3, Sep. 1997, pp. 1818–1825.

[13] Y. Yokokohji and T. Yoshikawa, "Bilateral control of master-slave manipulators for ideal kinesthetic coupling-formulation and experiment," *IEEE Trans. Robot. Autom.*, vol. 10, no. 5, pp. 605–620, Oct. 1994.

[14] I. Havoutis and S. Calinon, "Learning from demonstration for semi-autonomous teleoperation," *Auto. Robots*, vol. 43, no. 3, pp. 713–726, Mar. 2019.

[15] D. Lee and W. M. Spong, "Semi-autonomous teleoperation of multiple cooperative robots for human-robot lunar exploration," in *Proc. Spring Symp., Hum.-Robot Team Gone Before*, 2006, pp. 87–94.

[16] F. Chenf, B. Gao, M. Selvaggio, Z. Li, D. Caldwell, K. Kershaw, A. Masi, M. D. Castro, and R. Losito, "A framework of teleoperated and stereo vision guided mobile manipulation for industrial automation," in *Proc. IEEE Int. Conf. Mechatronics Automat.*, Harbin, China, Aug. 2016, pp. 1641–1648.

[17] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *Proc. Int. Conf. Robot. Automat.*, May 2017, pp. 2786–2793.

[18] D. Lee, "Semi-autonomous teleoperation of multiple wheeled mobile robots over the internet," in *Proc. Dyn. Syst. Control Conf.*, 2008, pp. 147–154.

[19] C. Ha, "Whole-body multi-modal semi-autonomous teleoperation of mobile manipulator systems," in *Proc. IEEE Int. Conf. Robot. Automat.*, May 2015, pp. 164–170.

[20] D.-H. Zhai and Y. Xia, "Adaptive control of semi-autonomous teleoperation system with asymmetric time-varying delays and input uncertainties," *IEEE Trans. Cybern.*, vol. 47, no. 11, pp. 3621–3633, Nov. 2017.

[21] K. Y. Lui, H. Cho, C. Ha, and D. Lee, "First-person view semi-autonomous teleoperation of cooperative wheeled mobile robots with visuo-haptic feedback," *Int. J. Robot. Res.*, vol. 36, nos. 5–7, pp. 840–860, Jun. 2017.

[22] M. U. Asad, U. Farooq, J. Gu, G. Abbas, R. Liu, and V. E. Balas, "A composite state convergence scheme for bilateral teleoperation systems," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 5, pp. 1166–1178, Sep. 2019.

[23] M. U. Asad, J. Gu, U. Farooq, E. V. Balas, Z. Chen, C. Chang, and A. Hanif, "A composite state convergence scheme for multilateral teleoperation systems," *Stud. Informat. Control*, vol. 30, no. 2, pp. 33–42, 2021.

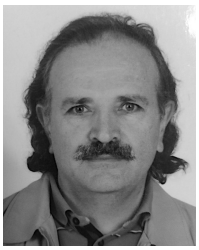
[24] W. S. Kim and S. P. Schenker, "Teleoperation training simulator with visual and kinesthetic-force virtual reality," *Proc. SPIE Hum. Vis., Vis. Process., Digit.*, vol. 1666, pp. 560–569, Aug. 1992.

[25] F. Niu and M. Abdel-Mottaleb, "HMM-based segmentation and recognition of human activities from video sequences," in *Proc. Int. Conf. Multimedia Expo*, 2005, pp. 804–807.

[26] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, "Action-conditional video prediction using deep networks in Atari games," in *Advances in Neural Information Processing Systems*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2015, pp. 2863–2871.
[27] M. Babaeizadeh, C. Finn, D. Erhan, R. H. Campbell, and S. Levine, "Stochastic variational video prediction," 2017, *arXiv:1710.11252*.
[28] A. X. Lee, R. Zhang, F. Ebert, P. Abbeel, C. Finn, and S. Levine, "Stochastic adversarial video prediction," 2018, *arXiv:1804.01523*.
[29] H. Coskun, S. Yucer, A. Akay, and Y. S. Akgul, "Learning based automated teleoperation system," in *Proc. 23nd Signal Process. Commun. Appl. Conf.*, May 2015, pp. 1240–1243.
[30] J. Liang, J. Mahler, M. Laskey, P. Li, and K. Goldberg, "Using dVRK teleoperation to facilitate deep learning of automation tasks for an industrial robot," in *Proc. Conf. Automat. Sci. Eng.*, Xi'an, China, Aug. 2017, pp. 1–8.
[31] J. A. Martijn Zeestraten, I. Havoutis, and S. Calinon, "Programming by demonstration for shared control with an application in teleoperation," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1848–1855, Jul. 2018.
[32] M. Pala and P. Sincak, "Towards the assisted teleoperation systems," in *Proc. Int. Congr. Ultra Mod. Telecommun. Control Syst.*, Oct. 2012, pp. 490–495.
[33] N. Hirose, A. Sadeghian, F. Xia, R. Martin-Martin, and S. Savarese, "VUNet: Dynamic scene view synthesis for traversability estimation using an RGB camera," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 2062–2069, Apr. 2019.
[34] N. Hirose, F. Xia, R. Martin-Martin, A. Sadeghian, and S. Savarese, "Deep visual MPC-policy learning for navigation," 2019, *arXiv:1903.02749*.
[35] F. Abi-Farraj, C. Pacchierotti, and P. R. Giordano, "User evaluation of a haptic-enabled shared-control approach for robotic telemanipulation," in *Proc. Int. Conf. Intell. Robots Syst.*, Oct. 2018, pp. 1–9.
[36] M. Selvaggio, J. Cacace, C. Pacchierotti, F. Ruggiero, and P. R. Giordano, "A shared-control teleoperation architecture for nonprehensile object transportation," *IEEE Trans. Robot.*, early access, Jun. 8, 2021, doi: 10.1109/TRO.2021.3086773.
[37] M. Hagenow, E. Senft, R. Radwin, M. Gleicher, B. Mutlu, and M. Zinn, "Corrective shared autonomy for addressing task variability," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 3720–3727, Apr. 2021.
[38] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," 2015, *arXiv:1511.05440*.
[39] H. Cai, C. Bai, Y.-W. Tai, and C.-K. Tang, "Deep video generation," in *Proc. ECCV*, 2018, pp. 366–382.
[40] C. Finn, I. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," in *Advance Neural Information Processing Systems*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2016, pp. 64–72.
[41] M. Sarkar, P. Pradhan, and D. Ghose, "Planning robot motion using deep visual prediction," 2019, *arXiv:1906.10182*.
[42] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, Nov. 2008.
[43] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, *arXiv:1301.3781*.
[44] H. Zhang, Z. Zhao, Y. Yu, K. Gui, X. Sheng, and X. Zhu, "A feasibility study on an intuitive teleoperation system combining IMU with sEMG sensors," in *Proc. ICIRA*, Newcastle, NSW, Australia, Aug. 2018, pp. 465–474.
[45] S. Mark Young and A. Neville Stanton, "Attention and automation: New perspectives on mental underload and performance," *Theor. Issues Ergonom. Sci.*, vol. 3, no. 2, pp. 178–194, Jan. 2002.
[46] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," in *Proc. 17th Int. Conf. Pattern Recognit.*, vol. 2, 2004, pp. 28–31.

ABDULLAH AKAY received the B.S. and M.S. degrees in computer engineering from Gebze Technical University (Gebze Institute of Technology), Kocaeli, Turkey, in 2010 and 2012, respectively, where he is currently pursuing the Ph.D. degree in computer engineering.

From 2011 to 2017, he was a Research Assistant at the Department of Computer Engineering, Gebze Technical University (Gebze Institute of Technology). He worked as a Research Scholar at the Department of Computer Science, University of Missouri, Columbia, MO, USA, from 2017 to 2018. His research interests include autonomous robots, semantic video and action prediction, structure from motion, and stereoscopic x-ray package inspection systems.



YUSUF SINAN AKGUL received the B.S. degree in computer engineering from Middle East Technical University, Ankara, Turkey, in 1992, and the M.S. and Ph.D. degrees in computer science from the University of Delaware, in 1995 and 2000, respectively. He worked as a Senior Vision Engineer with Cognex Corporation, Natick, MA, USA, from 2000 to 2005. He is currently with the Department of Computer Engineering, Gebze Technical University, Turkey. His research interests include the application of computer vision in medical image analysis, video analysis, object recognition, 3D analysis, and industrial inspection.