Proceeding of the IEEE
International Conference on Robotics and Biomimetics
Dali, China, December 2019

Programming by Visual Demonstration for Pick-and-Place Tasks using Robot Skills*

Peng Hao[1,2], Tao Lu[1], Yinghao Cai[1], and Shuo Wang[1,2,3]

*[1]The State Key Laboratory for Management and Control of Complex Systems,
Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
[2]University of Chinese Academy of Sciences, Beijing 100049, China
[3]Center for Excellence in Brain Science and Intelligence Technology,
Chinese Academy of Sciences, Shanghai 200031, China*

Email: {haopeng2017, tao.lu, yinghao.cai, shuo.wang}@ia.ac.cn

***Abstract* – In this paper, we present a vision-based robot programming system for pick-and-place tasks that can generate programs from human demonstrations. The system consists of a detection network and a program generation module. The detection network leverages convolutional pose machines to detect the keypoints of the objects. The network is trained in a simulation environment in which the train set is collected and auto-labeled. To bridge the gap between reality and simulation, we propose a design method of transform function for mapping a real image to synthesized style. Compared with the unmapped results, the Mean Absolute Error (MAE) of the model completely trained with synthesized images is reduced by 23% and the False Negative Rate FNR (FNR) of the model fine-tuned by the real images is reduced by 42.5% after mapping. The program generation module provides a human-readable program based on the detection results to reproduce a real-world demonstration, in which a long-short memory (LSM) is designed to integrate current and historical information. The system is tested in the real world with a UR5 robot on the task of stacking colored cubes in different orders.**

Keywords - Programming by Demonstration, Pick-and-Place, Sim-to-Real.

I. Introduction

Robots become an important part of our daily life and it is necessary for them to master basic skills like e.g. pick-and-place objects or adapt to new environments through easy programming. In past years, the expectation in using artificial intelligence to make the robot program to be easy has led many progresses in related technologies. Programming by Demonstration (PbD) is one of the powerful approaches to achieve the above goals. With demonstrations, the robots could generate the programs within less time to reproduce the demonstrated tasks.

There are several ways to gather demonstrations, such as Kinesthetic teaching [1], Teleoperation teaching [2] or Video teaching [3], et al. Kinesthetic and Teleoperation teaching require cumbersome human-robot interaction by professional equipments, and Video teaching uses cameras to record the demonstrations and does not require direct contact between humans and robots. Hence, Video teaching is a more user-friendly way to teach a robot.

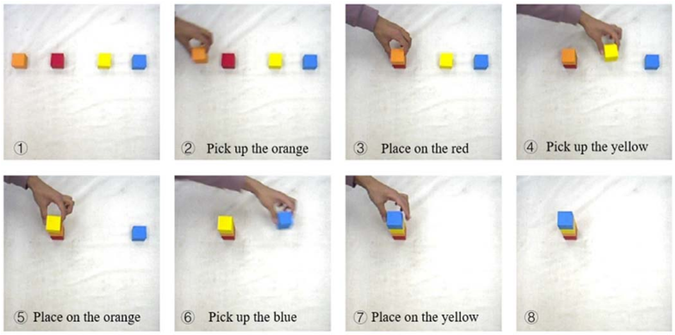

Fig. 1. The system generates programs from a human demonstration for pick-and-place tasks using robot skills.

Traditional PbD methods record the trajectories of the demonstrators [3], which has no consideration of understanding the behavioral intention and is hard to be reproduced in an unstructured environment. Using high-level instructions is another approach to simplify robot programming, e.g., semantically defined motion primitives, often called skills, which rely on a defined action and sensing sequence [4]. A skill-based robot program is human-readable and could easily generalize in an unstructured environment. Therefore, it is a promising way to introduce the parameterized skill representations into PbD to simplify robot programming of high-level tasks [5].

In this paper, we take a step in skill-based robot program by proposing a vision-based pick-and-place tasks programming system using robot skills. The system generates the program from a real-world demonstration and could reproduce the task in an unstructured environment. In order to illustrate the capabilities of the system, we tested the system with the task of stacking colored cubes in different orders, as illustrated in Fig. 1. In addition, our system can be used for other tasks related to pick-and-place such as cargo stacking, assembly, and sorting. This paper contains the following contributions:

1*)* A design method of transform function to bridge the reality gap. The detection network trained on the synthesized images can directly process the real images after mapping.

2) A long-short memory-based program generation algorithm. The information interaction between long memory and short memory robustly programs from demonstrations.

The remainder of this paper is organized as follows. In Section II, we introduce the related work of PbD and sim-to-real transformation. Section III reports the design method of

*This work was supported in part by the National Natural Science Foundation of China under Grant 61773378, U1713222, and U1806204; and in part by EPFF under Grant 61403120407.

978-1-7281-6321-5/19//$31.00 ©2019 IEEE

mapping function and the program generation algorithm. Section IV gives the results of the system on the task of stacking cubes. Finally, this paper is concluded in Section V.

II. Related Work

A. *Programming by Demonstration (PbD)*

PbD is an important research direction in robotics. Compared with learning by discovery and instruction, programming by demonstration can transfer complex skills to robots and does not have any professional skill requirements for the human demonstrator. Traditional programming by demonstration generally records the trajectories of the robot end-effector [6][7], which is hard to reproduce the task in a dynamic environment. Combining PbD with predefined skills is a new way to guarantee the generality of programming. Hsien-I Lin et al. [8] leverage deep neural networks to recognize human hand gestures in demonstrations and uses the Extensible Agent Behavior Specification Language (XABSL) language to program robotic pick-and-place tasks. Maj [5] introduces parameterized skill representations into PbD to simplify robot programming. Duan et al. [9] train a network in simulation by watching a user demonstration and then transfer it to a real robot. Tremblay et al. [10] propose a system that learns a human-readable program from a single demonstration in the real world and the learned program can be executed in the environment with different initial conditions.

B. *Sim-to-Real Transformation*

The difference between the simulated data and real data, known as the reality gap, forms the barrier to using simulated data on real robots. The computer vision community has devoted significant studies to the problem of adapting vision-based models trained in a source domain to a previously unseen target domain [11]. One way to reduce the reality gap is to use high quality rendered images, which has limited success on real robots [12]. Another way is using domain adaptation techniques to bridge the reality gap. Rusu et al. [13] explore using the progressive network architecture to adopt a model that is pre-trained on simulated pixels, and finds it has better sample efficiency than only fine-tuning or training in the real-world. Tobin [14] randomizes the simulator to expose the model to a wide range of environments at training time, then uses this method to train the detection network from the simulated data and deploy the detection network on a Fetch robot. However, this method requires a lot of simulation data in different lighting environments to get good generalization performance.

Our work draws inspiration from the work by Tremblay et al. [10]. We propose a vision-based programming system using primitive skills for pick-and-place tasks. Our system could transfer the models from simulation to real-world without large-scale train sets and robustly adapt to the dynamic environments.

III. Method

The flow chart of our system is shown in Fig. 2. First, the camera records a demonstration video and the position of objects in the scene are detected by the detection neural network. Then the state sequence which includes the position and the color of each object is fed to the program generation module to generate a human-readable program using robot skills. Finally, a robot will execute the program in a dynamic environment.

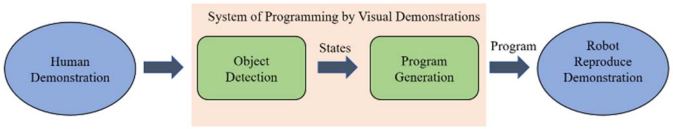
System of Programming by Visual Demonstrations
Human Demonstration
Object Detection
States
Program Generation
Program
Robot Reproduce Demonstration

Fig. 2. System overview.

A. *Object Detection and Sim-to-Real Transformation*

The object detection network infers the keypoints location of an object in each scene, which will be used as the input of the program generation module. The architecture of network utilizes the work of Cao et al. [15] which proposed a bottom-up detection method that enables real-time human keypoints detection. We modify it to detect the keypoints of the general object.

As shown in Fig. 3, the object detection network is divided into two phases, the first phase is the feature extraction network, and the second phase is the keypoints prediction network. The feature extraction network is the first ten layers of the pre-trained VGG-19 [16] on ImageNet. It ingests an image and outputs feature maps which are as the inputs of the keypoints prediction network. The keypoints prediction network is a two-branch multi-stage convolutional neural network (CNN). Branch 1 outputs confidence maps which include positions of all object keypoints in the image. Branch 2 outputs Part Affinity Fields (PAFs) which can be used to calculate the weight between each keypoint pair. A graph can be inferred from the outputs of the two branches, the nodes which are the keypoints and the edges are the weights of each keypoint pair. Detection results can be obtained by the classic graph cut algorithm.

We choose to use the L2 loss between the two-branch outputs and the ground-truth of the training data. Specifically, the loss functions of branch 1 and branch 2 at stage t are respectively:

$$f_C^t = \sum_{i=1}^{m} \|C_i^t - C_i^*\|_2^2 \tag{1}$$

$$f_P^t = \sum_{j=1}^{n} \left\|P_j^t - P_j^*\right\|_2^2 \tag{2}$$

Where C_i^* is the ground-truth of confidence map and P_j^* is the ground-truth of Part Affinity Fields. C_i^t, P_j^t are the outputs of branch 1 and branch 2, respectively. i, j are the indexes of the output feature maps for each branch, their total number is m, n. Let T be the number of stages included in phase 2, the overall objective is

$$f = \sum_{t=1}^{T} (f_C^t + f_P^t) \tag{3}$$

Because of the easy collection of training sets, the object detection network is trained on the synthesized images in a simulation environment. As Tobin analyzed in [14], due to the reality gap between the reality and the simulation, the detection network cannot directly deploy in the real world. As shown in Fig. 4, the neural network achieves good detection results on the synthesized images after training on a synthesized train set,

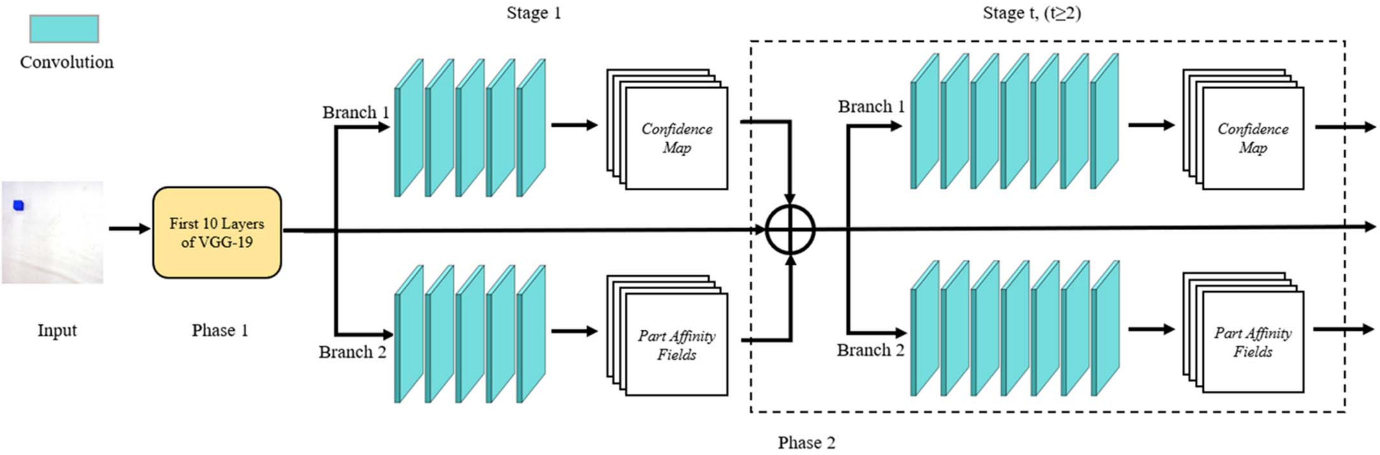

Fig. 3. The architecture of the two-branch multi-stage CNN. Each stage is divided into two branches, branch 1 is used to predict keypoints positions and branch 2 is used to calculate weight between each keypoint pair

while getting bad results on the real images. To overcome the reality gap, different from the domain randomization which needs large-scale train set to achieve good generalization performance, we propose a new method to transform real images to synthesized style using a designed mapping function.

In the scene with solid color background and objects, the color histogram of the image exhibits a multimodal distribution, where each peak represents one or more types of objects. As shown in Fig. 4, the high pixel portion represents the white background (RGB(255,255,255)) and the blue cube (RGB(0,0,255)), and the low pixel portion represents the blue cube. The multimodal distribution of the synthesized image is more pronounced than the real image. Based on this, we propose a design method of transform function to enhance the multimodal distribution of images containing solid color objects and backgrounds to achieve a real to synthetic style transformation. The method is as follows: First, determine the number n of colors in the scene and find the RGB values corresponding to each color. Second, select all or part of the above values as candidate attraction points. The principle of selection is to ensure that each color has a value in candidate attraction points. In this way, up to $3 \times n$ candidate attraction points can be obtained, and the final attraction points can be obtained after removing the repetition. Finally, add the function $f(x)$ in turn on the interval segmented by the attraction points.

$$f(x) = \begin{cases} g(x), t-l < x \le t \\ h(x), t < x \le t+r \end{cases} \tag{4}$$

In the interval $[t-l, t]$, $g(x)$ is a concave function which increases the pixel value to move closer to the attraction point t; Within the interval $[t, t+r]$, $h(x)$ is a convex function that decreases the pixel value to bring it closer to the attraction point t. l is half the distance between t and the previous attraction, r is half the distance between t and the next attraction. Connect all the piecewise functions $f(x)$ to get the final transformation function $T(x)$ which can be expressed as (5). Where n is the total number of $f(x)$, and the independent variables x and t are normalized. The transformation function $T(x)$ makes the multimodal distribution of real images more obvious and closer to the style of the synthesized image.

$$T(x) = \begin{cases} f_1(x), & 0 < x < t_1 + r \\ \vdots & \vdots \\ f_i(x), & t_i - l < x < t_i + r \\ \vdots & \vdots \\ f_n(x), & t_n - l < x < 1 \end{cases} \tag{5}$$

B. *Robot Skill Descriptions*

A robot skill is a fundamental software building block, that incorporates both sensing and action. The skill uses the current world state as input, along with a parameter which is provided at task programming time [17]. In this paper, we modified the skills in [17] to make it suitable for our system. The skills definition is in Table I. The **Parameter** is the data required for the execution of the skill. Specifically, Object A means the target needs to be picked up and Object B offers the position where Object A in gripper needs to place on. The world state needs to satisfy **Precondition** before the execution of the skill. The **Execution** is the sequence of action and sense included in the skill. The **Postcondition** is used to check whether the skill is finished.

TABLE I
PICK AND PLACE SKILLS

	Pick up	Place on
Parameter	Object A	Object B
Preconditions	Object A in world model, Gripper empty	Object B in world model, Object A in gripper
Execution	Detect the position of object A, Open gripper, Move to pre-grasp position, Move to grasp position, Close gripper, Move away	Detect the position of Object B, Move to pre-Place position, Move to Place position, Open gripper, Move away
Postconditions	Object A in gripper, Object A not at previous location	Gripper empty Object A at correct location

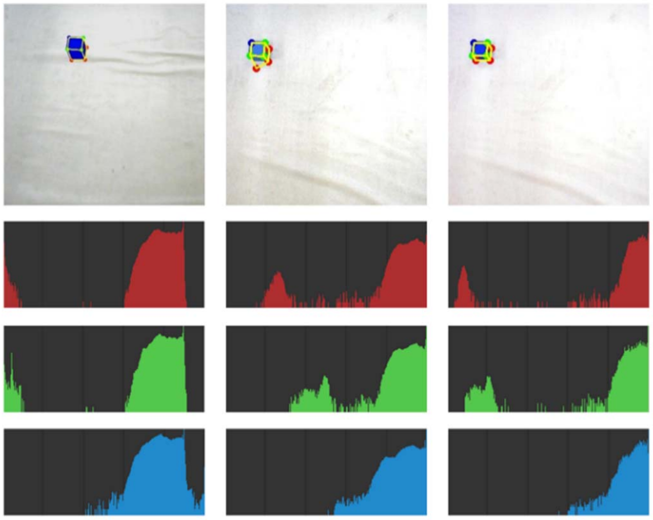

Fig. 4. From left to right are synthesized image, real image and real image after mapping. The following are histograms of each image RGB channels.

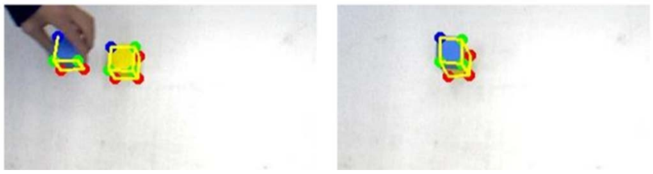

Fig. 5. The left picture shows the keypoints missed by the human hand occlusion. The right picture shows the detection network detecting two stacked cubes as one cube.

C. *Program Generation*

The input of the program generation is a state sequence of objects. Specifically, the state includes the position and color of each object, the number of objects. The output of the program generation module is a program written by the robot skills defined in Section III. B.

Tremblay [10] leverages the object detection network, relationship inference network and program generation network to process the demonstration video and generate a robot program. However, there may exist occlusion in the demonstration. As shown in Fig. 5, the detection network cannot detect the yellow cube if it is occluded, so the relationship inference network cannot work properly and a wrong program will eventually generate by the program generation network. Different from Tremblay's approach, we do not use neural networks to predict relationships or generate program. Instead, we propose a long-short memory-based algorithm that directly generates a robot program from the output of the detection network by analyzing the motion of the object in space during the demonstration.

The flow chart of the program generation algorithm is shown in Fig.6. The input is *State Sequence* which stores the positions and colors of the objects in each frame of teaching video. The first element of the *State Sequence* is used to initialize the long-memory *ObjectState* which stores the state of each object (grabbed or not). By querying the *ObjectState*, we can get a Boolean value *Picking* (T means that an object is being picked up, and F means that no object is being picked up). Short-memory *Checklist* is a FIFO that stores the states of adjacent n frames and is used to get the Boolean value *Moving* (T means **Pick up** may occur and F means **Place on** may occur). *Checklist* is updated by *State Sequence* until *State Sequence* is completely processed. As shown in the algorithm flow chart, the red line indicates that the short-memory avoids generating a wrong skill under the supervision of long-memory. For example, F in the green dotted box means that the short-memory thinks that **Place on** may occur. If the long-memory records an object has been picked up, the **Place on** will be generated, otherwise, **Place on** will not be generated. The same explanation applies to the T in the purple box. The blue flow indicates that if a **Pick up** or **Place on** occurs, the corresponding parameter **Object** is recorded and the *ObjectState* is updated. The information interaction between the two memories provides robustly programs. When all the State Sequence have been processed, the algorithm outputs the program Task in the form of **Pick up Object** and **Place on Object**.

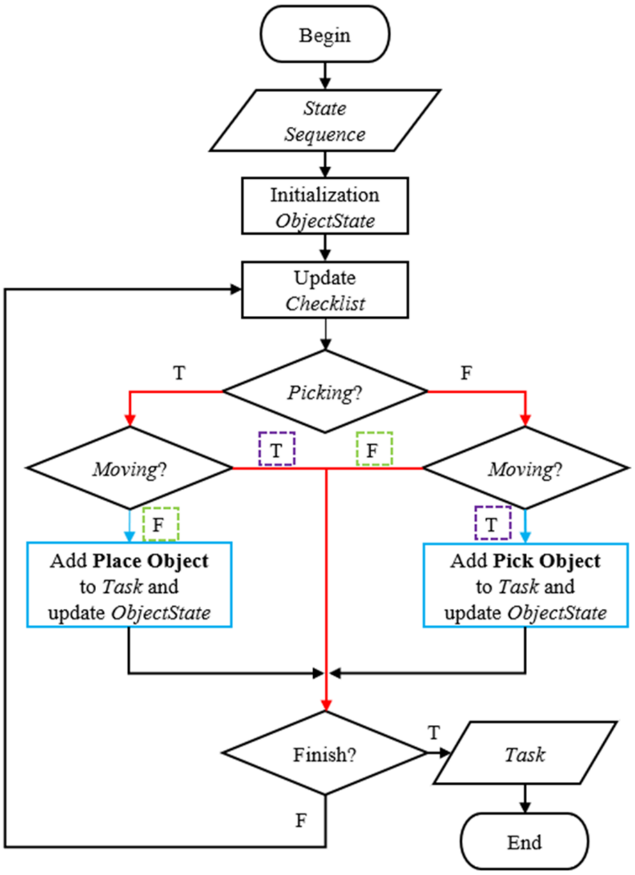
Begin
State Sequence
Initialization ObjectState
Update Checklist
T
Picking?
F
Moving?
T
F
Moving?
F
T
Add Place Object to Task and update ObjectState
Add Pick Object to Task and update ObjectState
Finish?
T
Task
F
End

Fig. 6. Flow chart of the program generation algorithm.

Our method does not predict relationships between every two objects, instead, we generate a robot program by analyzing the motion of the picked object. Therefore, if the occlusion does not affect the detection of the picked object, such as the occlusion in Fig.5, our system can generate the program correctly. In addition, the short-memory mechanism guarantees even if the detection is completely missing during picking, as long as the missing length is no more than *Checklist* length, our system can still generate the action correctly. This situation will be explained in Section IV. C.

IV. Experiment

In this section, we report tests of the individual components of the system followed by tests of the entire system with cube stacking task.

A. *Object Detection and Sim-to-Real Transformation*

We begin by assessing how accurately the object detection network which is trained on synthesized images can detect the location and pose of colored cubes in real images.

We built eight color cubes in V-rep [18] environment and eventually got about 3,000 cube images per color. We set the seven visible vertices on the cube as keypoints and designed an auto-label algorithm to realize the automatic annotation of the synthesized image. Finally, we got 18,000 effective annotation images for training the detection network. We also collect 2,000 images including human hands in the real world as negative samples in the train set.

We trained two detection network models. The model 1 was completely trained on the synthesized images, and the model 2 was the model 1 fine-tuned by the real images which include human hands. For testing, we collected 40 images that contain four color cubes that were in different poses. We manually labeled the cube vertices in these images as ground-truth.

We designed the mapping function for bridging the reality gap. First, based on the five colors composed in the image which are white, red, blue, yellow, and orange, we choose 0 and 255 which are the channel values of these colors as attraction points. Second, we choose the power function as the style of the transformation function. $g(x)$ is a root function and $h(x)$ is a quadratic square function. According to the method of Section III. A., We finally get the following mapping function $T(x)$ which is normalized and expressed as follows:

$$T(x) = \begin{cases} 4x^2 & , 0 < x \leq 0.5 \\ \sqrt{0.5x - 0.25} + 0.5 & , 0.5 < x \leq 1 \end{cases} \tag{6}$$

To measure the expected error in pixels, we use the mean absolute error (MAE) which can be expressed as:

$$\sum_{i=1}^{N} \frac{\delta_i}{N} \tag{7}$$

where N is the total number of vertices in all the images being used for evaluation, and δ_i is given by:

$$\delta_i = \sqrt{(u_i - \widehat{u_i})^2 + (v_i - \widehat{v_i})^2} \tag{8}$$

where u_i, v_i are the horizontal and vertical coordinates of the i-th vertex of the network prediction in the image coordinate, and $\widehat{u_i}$, $\widehat{v_i}$ are the ground-truth of the horizontal and vertical coordinates of the point in the image coordinate.

The results are shown in Table II where the false negative rate (FNR) is the percentage of cubes not detected. We mainly analyze the effect of transformation function on the detection results when the detection model is the same. As shown in Table II, whether it is model 1 or model 2, mapping the image with the function can reduce FNR and MAE. When the network is model 1, MAE is reduced by 23%. When the network is model 2, FNR is reduced by 42.5%. Some visual results are shown in Fig. 7. After mapping, the detection results have been greatly improved.

TABLE II
Result of the detection network

	MAE (pixels)	FNR (#/cubes)
Model 1 w/o Mapping	3.65	3/40
Model 1 w/ Mapping	**2.81**	**2/40**
Model 2 w/o Mapping	2.46	17/40
Model 2 w/ Mapping	**2.39**	**0/40**

B. *Program Generation*

In order to avoid redundant information processing, the demonstration videos are sampled at a fixed frequency. According to the function used to determine ***Moving***, the sampling frequency needs to meet the following requirements: it is low enough that when an object is picked up, the moving distance of the object between the adjacent two frames exceeds its own size; the frequency is high enough to guarantee the detection network outputs the same outputs in several adjacent frames when the object placed in a steady state. In this paper, the sampling frequency is 1.5Hz and the number of adjacent frames n is 3 which is also the length of the *Checklist*.

The program generation algorithm is evaluated by the percentage of correct generation from human demonstrations. We choose two cubes from four different color cubes and stack them in different orders. Finally, we collected twelve demonstration videos for testing program generation algorithm. Although we add occlusion in these demonstrations, our program generation algorithm can correctly generation the programs from the twelve videos using robot skills. Fig. 8 is one of the tests that shows our algorithm can generate correct programs of a pick-and-place task in the case of unstable detection.

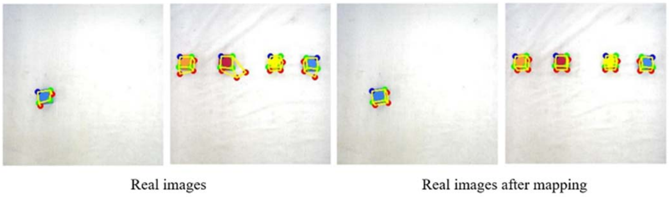

Fig. 7. The left column is the result of directly detecting real images and the right column is the result of detecting the same image after mapping.

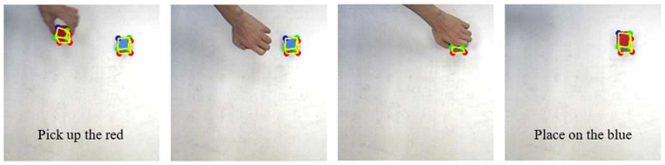

Fig. 8. Although the red cube is occluded by the human hand and the blue cube is occluded by the red cube, our algorithm can correctly generate programs.

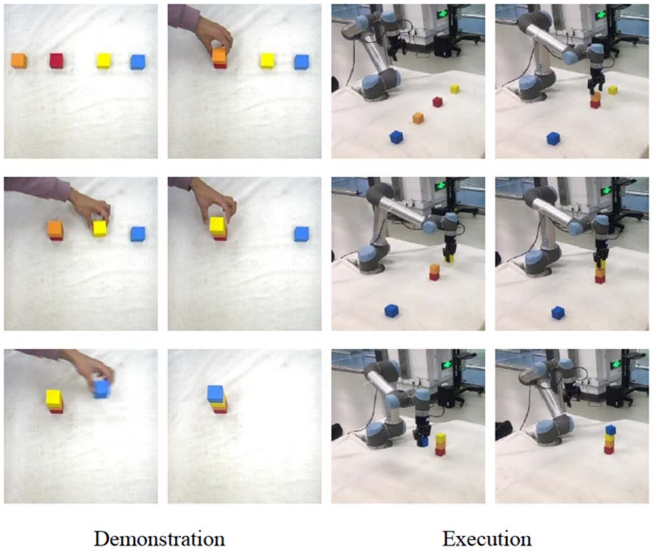

Fig. 9. The left column is the human demonstration, and the right column is the robot execution. The system generates a program from the human demonstration, and the system can control the robot to reproduce the demonstration even if the initial environment changes.

C. *Entire System*

The entire system is tested in our lab with a UR5 robot and an ASUS camera. In theory, the system has no limit on the number of actions included in the demonstration. In this paper, the task is to learn the stacking order of four cubes from human demonstrations and reproduce it in an unstructured environment. The process is as follows. First, a human teacher stacks the cubes in any order and the computer records the whole demonstration as videos using ROS camera nodes. Second, the detection network ingests the images of videos and outputs the state sequence. Third, the program generation module generates a program from the state sequence using robot skills. Finally, the robot reproduces the task in an unstructured environment after human confirm the program. The experiment process is shown in Fig. 9.

V. Conclusion

In this paper, we present a vision-based robot pick-and-place programming system that can generate programs from human demonstrations. Our system has two technical contributions. For object detection, we propose a design method of transformation function for mapping the real image to the synthesized style and improve the detection accuracy. For program generation, we propose a long-short memory (LSM) algorithm to generate the correct robot skill sequence even when the detection is unstable. We demonstrated effectiveness of the system on the task of stacking colored cubes in different orders. In addition, the system can be used for tasks related to robot pick-and-place such as cargo stacking, assembly, and sorting. There are also a lot of exciting directions for future work, such as using the detection information as the supervision information to train the transformation function, learning a policy from a demonstration, etc.

References

[1] Martin Tykal, Alberto Montebelli, and Ville Kyrki. "Incrementally assisted kinesthetic teaching for programming by demonstration." *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*, Christchurch, New Zealand, 2016, pp. 205-212.

[2] L. Peternel, T. Petrič and J. Babič, "Human-in-the-loop approach for teaching robot assembly tasks using impedance control interface," *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, 2015, pp. 1497-1502.

[3] Hsien-I Lin and Yu-Hsiang Lin, "A novel teaching system for industrial robots." *Sensors*, vol. 14, no. 4, pp. 6012-6031, March 2014.

[4] M. R. Pedersen, L. Nalpantidis, R. S. Andersen, C. Schou, S. Bøgh, V. Krüger, et al. "Robot skills for manufacturing: From concept to industrial deployment." *Robotics and Computer-Integrated Manufacturing*, vol.37, pp. 282-291, February 2016.

[5] M. Stenmark. and E. A. Topp, "From demonstrations to skills for high-level programming of industrial robots." *AAAI Fall Symposium-Technical Report*, Phoenix, Arizona, 2016.

[6] M. F. Zaeh and W. Vogl, "Interactive laser-projection for programming industrial robots," *2006 IEEE/ACM International Symposium on Mixed and Augmented Reality*, Santa Barbard, CA, 2006, pp. 125-128.

[7] J. Lambrecht, H. Walzel and J. Krüger, "Robust finger gesture recognition on handheld devices for spatial programming of industrial robots," *2013 IEEE RO-MAN*, Gyeongju, 2013, pp. 99-106.

[8] Hsien-I Lin and Y. P. Chiang, "Understanding Human Hand Gestures for Learning Robot Pick-and-Place Tasks." *International Journal of Advanced Robotic Systems*, vol. 12, no. 5, May 2015.

[9] Y. Duan, M. Andrychowicz, B. Stadie, J. Ho, J. Schneider, I. Sutskever, et al. "One-shot imitation learning," *Neural Information Processing Systems (NIPS)*, Long Beach, California, 2017, pp. 1087-1098.

[10] J. Tremblay, T. To, A. Molchanov, S. Tyree, J. Kautz, and S. Birchfield, "Synthetically Trained Neural Networks for Learning Human-Readable Plans from Real-World Demonstrations," *IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, QLD, 2018, pp. 1-5.

[11] J. Hoffman, S. Guadarrama, E. Tzeng, R. Hu, J. Donahue, R. B. Girshick, et al. "Lsda: Large scale detection through adaptation." *Neural Information Processing Systems (NIPS)*, Montreal, Quebec, 2014, pp. 3536-3544.

[12] S. James and E. Johns, "3d simulation for robot arm control with deep q-learning." *In NIPS 2016 Workshop: Deep Learning for Action and Interaction*, Barcelona, Spain, 2016.

[13] Andrei A. Rusu, M. Vecerik, T. Rothorl, N. Heess, R. Pascanu, and R. Hadsell. "Sim-to-real robot learning from pixels with progressive nets. " *Proceedings of the 1st Annual Conference on Robot Learning*, Mountain View, California, 2017.

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

[15] Z. Cao, T. Simon, S. Wei, and Y. Sheikh, "Realtime Multi-person 2D Pose Estimation Using Part Affinity Fields," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, Hawaii, 2017, pp. 1302-1310.

[16] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations (ICLR)*, San Diego, CA, 2015.

[17] M. R. Pedersen, L. Nalpantidis, A. Bobick, and V. Krüger, "On the integration of hardware-abstracted robot skills for use in industrial scenarios," *Second International IROS Workshop on Cognitive Robotics Systems: Replicating Human Actions and Activities*, Tokyo, Japan, 2013.

[18] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," 2013 *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, 2013, pp.1321-1326.