# Learning from Observation-Only Demonstration for Task-Oriented Language Grounding via Self-Examination

**Tsu-Jui Fu**[*]
University of California, Santa Barbara

**Yuta Tsuboi**
Preferred Networks, Inc.

**Yuta Kikuchi**
Preferred Networks, Inc.

**Sosuke Kobayashi**
Preferred Networks, Inc.

## Abstract

Imitation learning is an effective method for learning a control policy from expert demonstrations. Combining imitation with natural language instruction promises to further make imitation learning more flexible and useful in real-world applications. However, most existing imitation methods rely on the assumption demonstrations also contain action sequences and that the agent can interact with them to explore alternative trajectories in the space, which greatly limits the practicality of such methods. We focus on imitation learning with observation-only language-conditional demonstrations in which ground truth action sequences are not explicitly given. We propose a method which initially pre-trains modules to capture the inverse dynamics of the world and learns how to describe the demonstration in natural language. In a second phase, these modules are used to generate additional training instances which can be explored self-examination. We evaluate our method on pick-and-place tasks show that the self-examination improves language grounding.

## 1 Introduction

Humans are able to learn quickly in new environments by observing others. Different from conventional imitation learning [2, 12], imitation from observation (IFO) [11, 18, 19] and instruction-conditioned imitation learning [1, 7] train agents similar to the way how humans learn.

IFO provides human-like imitation in which the raw sensor observations of expert behaviors are used but the executed actions by the expert, such as the trajectory of their joint angles, are not used. This enables learning from a large number of existing resources such as videos of humans performing tasks. To address IFO problem, behavioral cloning from observation (BCO) [17] trains an inverse dynamic model to infer the missing actions and learn a policy using the estimated actions.

From another point of view, humans can communicate their objectives to others by not only demonstrating but also describing them. Most of the previous imitation learning tasks are specified only by goal images or video frames [8, 14]. Since the imitation task is visually described, we need a lot of demonstrations to describe tasks. For example, many images are required to tell whether shape matters or color matters. To overcome this limitation, instruction-conditioned visual tasks have been proposed in which the task of a demonstration is also described by natural language instructions. Since instructions can express the task in a more abstract way, we expect agents are generalized so that they behave better under unknown situations. Adversarial Goal-Induced Learning from Examples

[*]The work was performed during an internship at Preferred Networks, Inc.

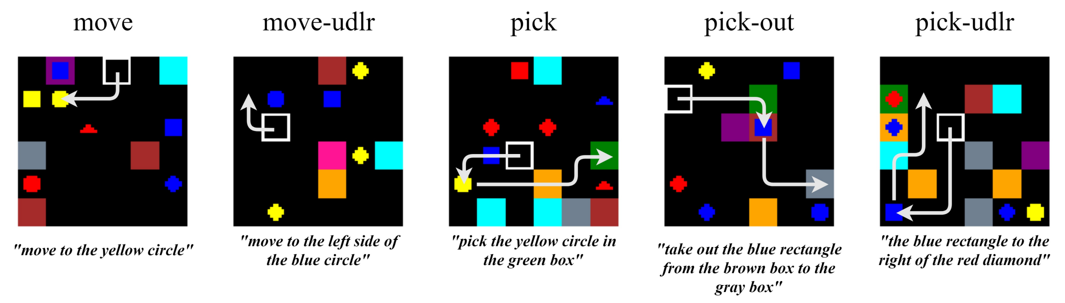
move
move-udlr
pick
pick-out
pick-udlr
"move to the yellow circle"
"move to the left side of the blue circle"
"pick the yellow circle in the green box"
"take out the blue rectangle from the brown box to the gray box"
"the blue rectangle to the right of the red diamond"

Figure 1: Example of 5 different tasks. The white squares depict robot arms and the arrows depict the trajectories of expert demonstrations.

(AGILE) [1] and Language-Conditioned Reward Learning (LC-RL) [7] have been proposed to learn a reward function from the instruction-conditioned demonstrations and use it to train policies.

Both of them have desirable properties for real-world applications. However, both AGILE and LC-RL assume expert actions are given and have to interact with demonstrations, i.e., for the sampling of negative examples, actions different from expert actions have to be performed under the demonstration's state. In other words, the demonstrations must be collected in a virtual environment, i.e. a simulation, in practice, but it limits their application field. Instead of interacting with demonstrations, we introduce a two-phase method to train policies in a simulator, as illustrated in Fig. 2. In the 1st pre-training phase, we train 1) *inverse module* which estimates actions of demonstration states, 2) *instruction module* which selects an appropriate instruction for states not observed in the demonstrations, and 3) *reward module* which estimates a reward during a simulation. Actions estimated by the inverse module are used for not only the training of the reward module but also the pre-training of a policy. In the 2nd-phase, we further improve the policy in the simulation. The instruction module selects a pseudo instruction for the initial state and a reward module gives reward feedback to actions performed by the policy, what we call *self-examination*. In this way, we can try-and-error under visual-language setting of observation-only imitation learning.

## 2 Proposed Tasks and Environment

We build an environment for pick-and-place tasks where we have to move the robot arm to a specific position or pick an object into the correct place which is specified by an instruction. We propose 5 tasks, *move*, *move-udlr*, *pick*, *pick-out*, and *pick-udlr* for different situations of pick-and-place problems, as illustrated in Fig. 1. For more detail, please see the Appendix. A. Let a demonstration $D$ be a pair of a task instruction $t$ and consecutive state images, $(t, \{s_1, s_2, \ldots\})$. Our visual-language grounding problem is learning an action policy, $\phi$, using both $N$ demonstrations $\{D_i\}_{i=1}^{N}$ and a simulator which simulates the forward dynamics of the environment, $s_k = \mathrm{sim}\,(s_{k-1}, a)$. We also assume random initial states $s_1$ can be generated. Not that we assume no visual gap between simulations and real environments in this work and leave the filling the gap [3, 9, 16] for future work.

## 3 Methodology

The overview of our proposed method is illustrated in Fig. 2.

### 3.1 Modules

To enable self-examination in the simulator, we train the instruction module, reword module, and policy using expert demonstrations at the pre-training phase (the left part of Fig. 2). Although expert actions are required for the training of the reward module and the policy, they are not observable. Therefore, at first, the inverse module is trained to estimate those missing actions in the demonstrations.

**Inverse Module**: We first collect numerous state-action pairs $(s_i, s_{i+1}, a_i)$ by randomly performing actions in the simulator. Using these pairs, we train a inverse module which predicts an action for

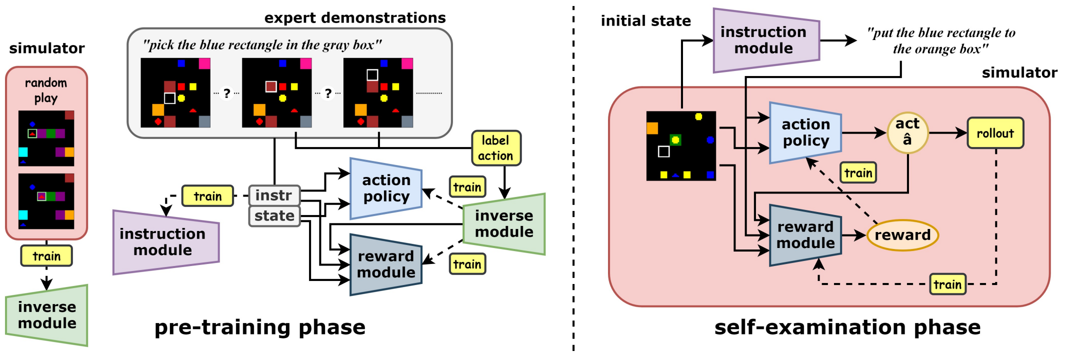

Figure 2: The overview architecture and training flow of both pre-training and self-examination.

realizing a transition between two consecutive states $(s_i, s_{i+1})$:

$$a' = \mathrm{inv}(s_i, s_{i+1}),$$

in the same way as a supervised classification problem. After the training, the unobserved actions in the demonstrations can be estimated using this inverse module.

**Instruction Module**: Since language instructions are not available in the simulation, for an initial state, the instruction module $\mathrm{ins}$ selects a suitable instruction from an instruction pool:

$$\hat{t} = \mathrm{argmax}_{\tilde{t} \in P}\, \mathrm{ins}\left(s, \tilde{t}\right),$$

where $\mathrm{ins}(s, t) \in [0, 1]$ is a ranking function and $P = \{t_i\}_{i=1}^{N}$ is the instruction pool which consists of $N$ instructions in the demonstrations $D$. [2] $\mathrm{ins}$ is trained as a binary classifier which discriminates the whether the correct state-instruction pair or not. For the training of $\mathrm{ins}$, we use the state-instruction pairs in the demonstrations as the positive pairs and the different instructions chosen randomly as the negative pairs. At the inference time, if the confidence is low, $\mathrm{ins}(s, \hat{t}) \leq 0.8$, for the selected instruction $\hat{t}$, then we skip the state, and generate a new initial state again.

**Reward Module**: A reward module $\mathrm{rwd}$ gives a reward feedback to the policy in the simulation:

$$\hat{r} = \mathrm{rwd}(s, t, a) \in [0, 1].$$

$\mathrm{rwd}$ is also trained as a binary classifier, and we consider the actions labeled by the inverse module $\mathrm{inv}$ in the demonstrations as the positive cases and randomly different actions as negative cases during the pre-training.

**Action Policy**: A policy maps current state $s$ and instruction $t$ to next action:

$$\hat{a} = \phi(s, t).$$

At the pre-training phase, behavioral cloning is employed to train a policy using demonstrations and the labeled action $a'$ which is estimated by the inverse module.

### 3.2 Self-Examination

With the instruction module and the reward module, we can further improve the policy $\phi$ in the simulator (the right part of Fig. 2).

Firstly, we randomly generate an initial state $s_1$ and select its instruction $\hat{t}$ by the instruction module. Then, using the simulator, the policy repeat the rollout from current state to next state by predicting a next action. After these rollouts, we apply the reward module to give reward feedback for each step of the execution trajectory: $T = \{(s_1, \hat{t}, \hat{a}_1, \hat{r}_1), (s_2, \hat{t}, \hat{a}_2, \hat{r}_2), (s_3, \hat{t}, \hat{a}_3, \hat{r}_3)...\}$. Finally, we update the policy $\phi$ via Policy Gradient method [15] using $T$. The reward module is also updated at this stage by adding the actions the policy performed to the negative examples. In the this process, the policy $\phi$ can try-and-error under different states and different instructions without interacting with demonstrations themselves. Algorithm 1 describes the detail of the self-examination procedure.

[2]Note that, since our preliminary experiment shows learning instruction generation is not robust, we employ the retrieval method instead of instruction generation.

**Algorithm 1** self-examination phase

```
$\phi$: action policy
$\text{ins}$: instruction module
$\text{rwd}$: reward module

while self-examination do
    $s_1 \leftarrow$ a random initial state
    $\hat{t} \leftarrow$ select a suitable instruction by $\text{ins}$

    $\{(s_1, \hat{t}, \hat{a}_1), (s_2, \hat{t}, \hat{a}_2), \ldots\} \leftarrow$ adopt $\phi$ to rollout $(s_1, \hat{t})$ on the simulator
    $\{(s_1, \hat{t}, \hat{a}_1, \hat{r}_1), (s_2, \hat{t}, \hat{a}_2, \hat{r}_2), ...\} \leftarrow$ the reward module $\text{rwd}$ gives reward

    update $\phi$ via Policy Gradient
    update $\text{rwd}$ using $\{(s_1, \hat{t}, \hat{a}_1), (s_2, \hat{t}, \hat{a}_2), \ldots\}$ as negative cases
end while
```

| Task | baseline | self-exam. [5K] | self-exam. [10K] |
|---|---|---|---|
| move | 76.6% | **79.4% (+2.8)** | 78.4% (+1.8) |
| move-udlr | 69.0% | **73.4% (+4.4)** | 72.4% (+3.4) |
| pick | 13.0% | 15.0% (+2.0) | **15.4% (+2.4)** |
| pick-out | 32.8% | 34.0% (+1.2) | **34.6% (+1.8)** |
| pick-udlr | 22.6% | **26.0% (+3.4)** | 24.8% (+2.2) |

Table 1: The success rate under **normal** setting.

| baseline | self-exam. |
|---|---|
| 62.8% | **68.4% (+5.6)** |
| 56.8% | **60.6% (+3.8)** |
| 11.2% | **14.6% (+3.4)** |
| 27.4% | **29.8% (+2.4)** |
| 19.8% | **21.0% (+1.2)** |

Table 2: **zero-shot** setting.

## 4 Implementation Detail

We employ gated-attention (GA) [7] to encode the pair of a state image $s$ and a instruction $t$ as follows:

$$\text{GA}(s, t) = \text{CNN}(s) \odot h(\text{GRU}(t)),$$

where CNN extracts the visual feature of the state image, GRU models the feature of the input instruction, and $h$ is a linear layer with sigmoid activation. $h$ will project and extend the GRU feature into the same shape as the CNN feature, and $\odot$ represents the Hadamard product. And the feature from GA will pass different multilayer perceptrons (MLPs) for each purpose of the action policy $\phi$, the instruction module $\text{ins}$, and the reward module $\text{rwd}$.

We apply a 4-layer convolutional neural network (CNN) with kernel size 3, feature size 32, stride 2, and padding 1 to extract the visual feature of the state image $s$. We adopt bidirectional Gated Recurrent Units (bi-GRU) [6] with hidden size of 64 to model the input instruction. The word embedding with size 8 is randomly initialized for bi-GRU, then trained with the whole network. During training, we set the dropout rate 0.25, the learning rate of the pre-training 8e-4, and the learning rate of self-examination 3e-5. We train our method using Adam optimizer [10] and implement it under PyTorch [13].

## 5 Experimental Results

### 5.1 Experimental Settings

We evaluate our proposed method on *move*, *move-udlr*, *pick*, *pick-out*, and *pick-udlr* tasks, which are described in Section 2. There are 80K demonstrations for each task in the pre-training phase. For the self-examination phase, we randomly generate up to 10K of initial states of which the instruction module can find the instructions. Implementation detail is shown in Appendix. 4. For the baseline, we use the pre-trained policy in the 1st phase, since it is trained as the same way as behavioral cloning from observation (BCO) [17], which is the state-of-the-art method of imitation from observation.

### 5.2 Quantitative Results

Table. 1 shows the success rate of both the baseline and the proposed method on 500 evaluation data for each task. For the baseline BCO, it achieves 76.6% and 69.0% for simpler task, *move*

| Task | Suitable Rate |
|---|---|
| move | 91.725% |
| move-udlr | 81.145% |
| pick | 90.190% |
| pick-out | 77.940% |
| pick-udlr | 76.810% |

Table 3: Suitable rate of the instruction module

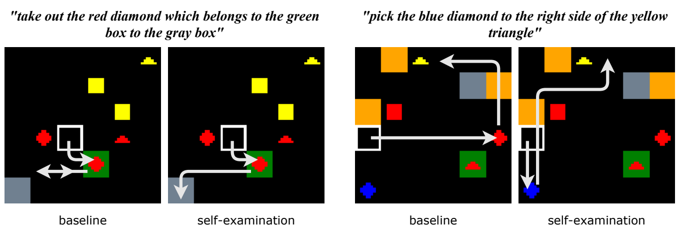
"take out the red diamond which belongs to the green box to the gray box"
"pick the blue diamond to the right side of the yellow triangle"
baseline
self-examination
baseline
self-examination

Figure 3: Case study.

and *move-udlr*, and 13.0%, 32.8%, and 22.6% for more difficult task, *pick*, *pick-out*, and *pick-udlr*, respectively. For the proposed method, we used 5K or 10K state-instruction pairs for exploration. The results of *self-exam.* in Table. 1 show that no matter 5K or 10K, the self-examination phase improves 1.2-4.4 points of success rate which means exploration in the simulation actually benefits to all tasks.

### 5.3 Detailed Analysis

**Zero-shot Generalization**: To investigate the generability of visual-language grounding, we evaluate under zero-shot setting where new combinations of attribute-object pairs are unseen in the training instructions [4]. For example, there are *red circle* and *yellow rectangle* in the training but contains *yellow circle* in the testing instructions. To evaluate the success rate for unseen attribute-object pairs, we also use 80K demonstrations for the pre-training and 5K initial states for the self-examination.

Table. 2 shows the results. Even under the zero-shot setting, we can see that there is only little performance drop and the exploration in the self-examination can improve 1.2-5.6 points of success rate over the baseline.

**Selected Instructions Suitability**: To make sure the instructions selected by ins are suitable for the initial state generated by the simulator, we evaluate the suitable rate of state-instruction pairs.

As shown in Table. 3, the suitable rate for all tasks is larger than 76% which means the instruction module actually selects suitable instructions for most of the initial states, making the exploration possible in the simulation environment.

**Case Study**: The executions in the comparison between the baseline and the self-examination are shown in Fig. 3. Since the baseline only sees the perfect actions in the training, it leads to be stuck and go back and forth repeatedly under new situations (the left case), although the proposed method can explore under the situation not in the demonstrations. In addition, the self-examination can recover the situation in which the baseline chose the wrong target object or the wrong target position for some cases (the right case). Appendix. C demonstrates more case studies.

## 6 Conclusion and Future Work

We address imitation learning problem from observation-only demonstrations without ground-truth action under visual-language setting. We propose a two-phase method which enables self-examination in the simulation. By trained using expert demonstrations, our method can provide instructions and reword feedback for situations not in expert demonstrations. The experimental results show that the proposed method improves the success rates of several pick-and-place tasks. Although we validate our method on the virtual environment, further study is needed for real-world applications, such as voice-control robot learning via human demonstrations.

## Acknowledgement

We appreciate the feedback offered by Jason Naradowsky.

## References

[1] D. Bahdanau, F. Hill, J. Leike, E. Hughes, A. Hosseini, P. Kohli, and E. Grefenstette. Learning to understand goal specifications by modelling reward. In *ICLR*, 2019.

[2] M. Bain and C. Sammut. A framework for behavioural cloning. In *Machine Intelligence*, 1995.

[3] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. P. Sampedro, K. Konolige, S. Levine, and V. Vanhoucke. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *ICRA*, 2018.

[4] D. S. Chaplot, K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In *AAAI*, 2018.

[5] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. BabyAI: first steps towards grounded language learning with a human in the loop. In *ICLR*, 2019.

[6] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS Workshop*, 2014.

[7] J. Fu, A. Korattikara, S. Levine, and S. Guadarrama. From language to goals: Inverse reinforcement learning for vision-based instruction following. In *ICLR*, 2019.

[8] J. Ho and S. Ermon. Generative adversarial imitation learning. In *NIPS*, 2016.

[9] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *CVPR*, 2019.

[10] D. P. Kingma and J. Ba. Adam: a method for stochastic optimization. In *ICLR*, 2015.

[11] Y. Liu, A. Gupta, P. Abbeel, and S. Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *ICRA*, 2018.

[12] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *ICML*, 2000.

[13] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[14] D. Pathak, P. Mahmoudieh, G. Luo, P. Agrawal, D. Chen, Y. Shentu, E. Shelhamer, J. Malik, A. A. Efros, and T. Darrell. Zero-shot visual imitation. In *ICLR*, 2018.

[15] R. S. Sutton, D. McAllester, N. S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999.

[16] 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. In *IROS*, 2017.

[17] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. In *IJCAI*, 2018.

[18] F. Torabi, G. Warnell, and P. Stone1. Recent advances in imitation learning from observation. In *IJCAI*, 2019.

[19] T. Yu, C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. In *RSS*, 2018.

## A Environment for Proposed Task

**Environment**: There are 6 actions in our environment:

- U/R/D/L: move the arm up/right/down/left
- P: pick or place the object (pick if the arm is empty, otherwise place the holding object)
- S: stop to indicate the end of the task described by the instruction

As Fig. 1, the environment is a 6x6 observable 2D grid-word which is populated with objects of different shapes or colors and boxes (those grids with all color filled in) of different colors. The specification is as following:

- Object Shape: *circle*, *rectangle*, *diamond*, *triangle*
- Object Color: *red*, *yellow*, *blue*
- Box Color: *orange*, *green*, *purple*, *pink*, *brown*, *gray*, *cyan*

All the objects and boxes are randomly placed in the grid-world.

During the evaluation, we check the final state when the stop action S is predicted or the maximum number (40) of action is exceeded. We believe it is important that the agent knows when to finish. Although there can be a probable case that agent achieves the target during the moving process but finally moves out from the target, we see those cases as failures.

**Task**: We propose 5 tasks, *move*, *move-udlr*, *pick*, *pick-out*, and *pick-udlr* for different situations of pick-and-place problems.

- *move*: move the robot arm to the object
- *move-udlr*: move the robot arm to the up/down/left/right side of the object
- *pick*: pick the object into the box
- *pick-out*: take out the object from the box into another box
- *pick-udlr*: pick the object to the up/down/left/right side of another object

As BabyAI [5], the related instructions are generated by templates (1 for *move*, 8 for *move-udlr*, 12 for *pick*, 36 for *pick-out*, 24 for *pick-udlr*) with different object shapes, object colors, and box colors. Note that our method does not require to access to these templates and they only used for the generation of the demonstrations. And the corresponding ground-truth demonstrations are also automatically produced where all paths are the shortest one. To avoid the ambiguity, we make sure that there is only one valid object or box mentioned in the instruction. For example of *pick-out* task in Fig. 1, there are two blue rectangles in the environment but only one in the brown box, therefore, it will not cause any ambiguity.

## B Self-Examination

## C Case Study

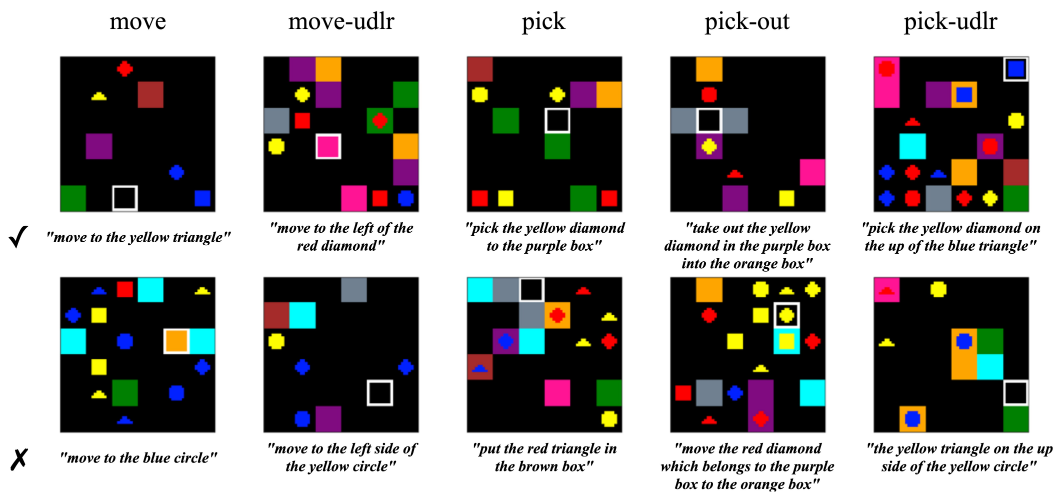
move
move-udlr
pick
pick-out
pick-udlr
√
"move to the yellow triangle"
"move to the left of the red diamond"
"pick the yellow diamond to the purple box"
"take out the yellow diamond in the purple box into the orange box"
"pick the yellow diamond on the up of the blue triangle"
✗
"move to the blue circle"
"move to the left side of the yellow circle"
"put the red triangle in the brown box"
"move the red diamond which belongs to the purple box to the orange box"
"the yellow triangle on the up side of the yellow circle"

Figure 4: Case study of the instructions selected by the instruction modulesins.

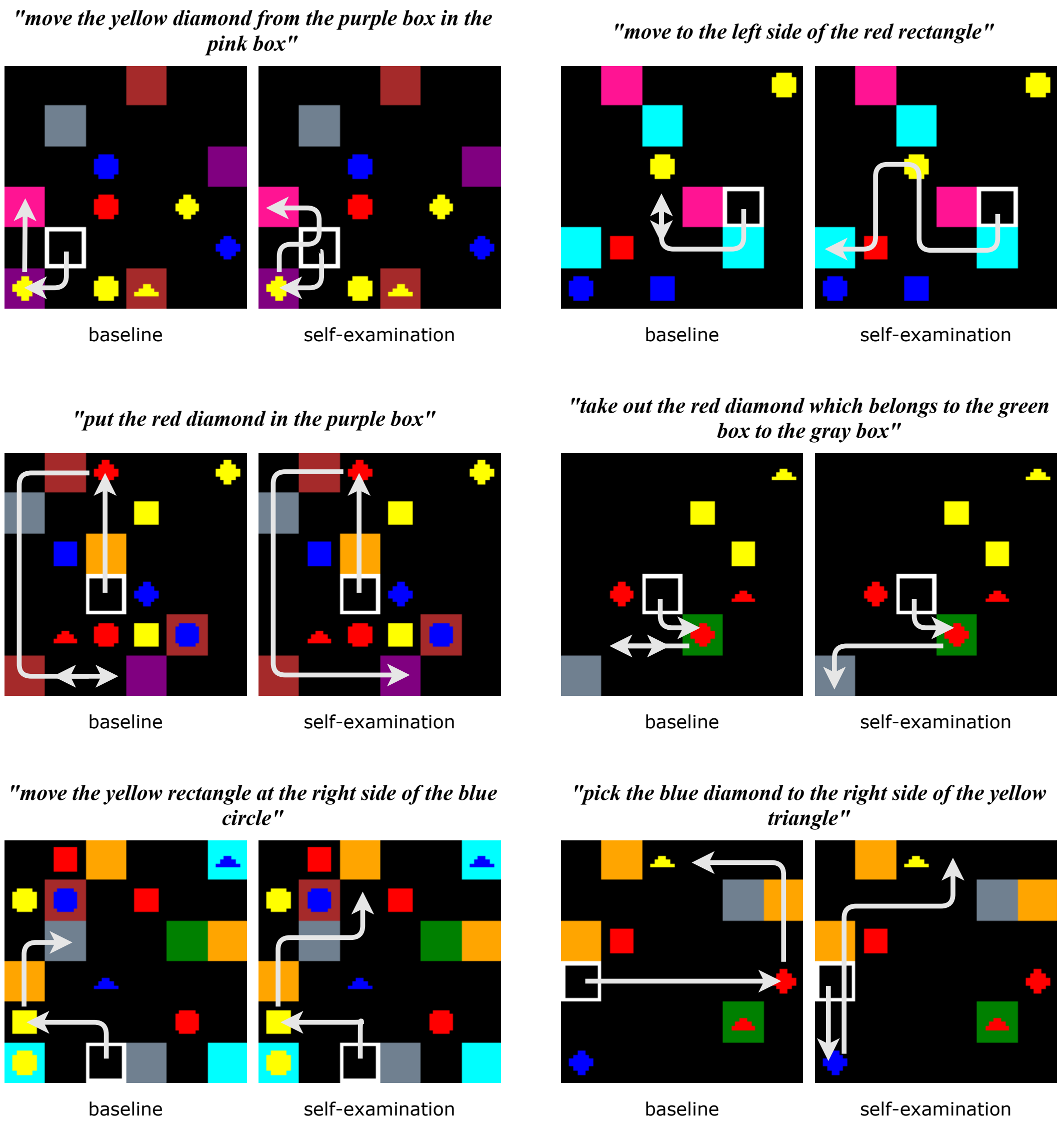
"move the yellow diamond from the purple box in the pink box"
baseline
self-examination
"move to the left side of the red rectangle"
baseline
self-examination
"put the red diamond in the purple box"
baseline
self-examination
"take out the red diamond which belongs to the green box to the gray box"
baseline
self-examination
"move the yellow rectangle at the right side of the blue circle"
baseline
self-examination
"pick the blue diamond to the right side of the yellow triangle"
baseline
self-examination

Figure 5: Examples of the different trajectory between the baseline and self-examination.