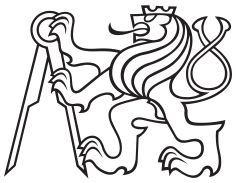
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Robotic Reaching of Objects Based on Verbal Description and Human Gesture in MyGym Simulator

Egor Sarana

Supervisor: Mgr. Michal Vavrečka, Ph.D.
June 2023

Acknowledgements

I would like to express my deepest thanks to my supervisor Mgr. Michal Vavrečka, Ph.D. Mr. Vavrečka advised me whenever I ran into trouble or had a question about the simulator.

I am also grateful to the CTU for all the knowledge I acquired during my study.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of the university theses.

Prague, 02. June 2023

Abstract

Interaction between humans and robots has been an ongoing research topic for decades. There are many types of such interaction, but this work is focused on the interaction based on verbal description and human gestures. This text describes the implementation of the human module in myGym simulator, new Reach gesture task, and the language module. Finally, we train new tasks with different robots, different reinforcement learning algorithms and evaluate them.

Keywords: robotic reaching, human gestures, language module

Supervisor: Mgr. Michal Vavrečka, Ph.D.

Abstrakt

Interakce mezi lidmi a roboty je předmětem výzkumu po celá desetiletí. Existuje mnoho typů takové interakce, ale tato práce je zaměřena na interakci založenou na verbálním popisu a lidských gestech. Tento text popisuje implementaci lidského modulu v simulátoru myGym, novou úlohu Reach gesture a jazykový modul. Nakonec trénujeme nové úlohy s různými roboty, různými algoritmy posilovaného učení a vyhodnocujeme je.

Klíčová slova: robotické dosahování, lidská gesta, jazykový modul

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Introduction

The original purpose of the emergence of robots is to serve people and help make their lives better. Therefore, as they have been developing, the importance of robot-human interaction (HRI) has only grown over time. HRI has important applications in the industrial field. While humans have the flexibility and the intelligence to consider different approaches to solving a problem, choose the best option from all the options, and then command robots to complete tasks, robots can be more accurate and consistent in performing repetitive and dangerous work.

Another important area is healthcare. In the last few decades, the idea of how a human and a robot interact with each other is one of the factors that has been widely considered in the development of rehabilitation robots. For example, HRI plays a vital role in the development of exoskeleton rehabilitation robots because the exoskeleton system is in direct contact with the human body.

myGym Simulator focuses on robot training, so HRI is unidirectional here. It has a workspace with a human, but there is no interaction. In this work, we will focus on the simplest types of interaction - understanding a human gesture and simple expressions in natural language.

An excellent job of gesture recognition has been done in the work [RSKP10]. Gesture recognition occurs in real time using the camera. The image from the camera is converted to black and white using global threshholding. Then the object of interest is cropped - the human hand (the figure 1.1). Next comes the removal of parts that are not related to gestures, for example, the

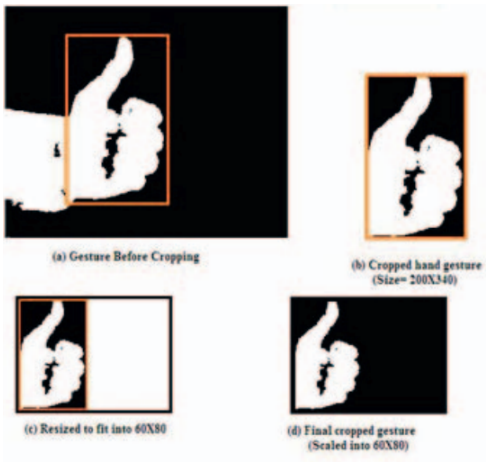


Figure 1.1: Scaling cropped hand gesture to fit into desired dimension.

arm and wrist. Then pattern matching with PCA takes place. It is faster than neural networks and helps to cope with a lot of noise (pixels that do not carry information). However, we will focus on a more straightforward task and only one gesture.

1.2 Materials and methods

In this work, version 2.1.2 of myGym simulator [VSM+20] is used. myGym is a toolkit suitable for the fast prototyping of neural networks in the area of robotic manipulation and navigation.

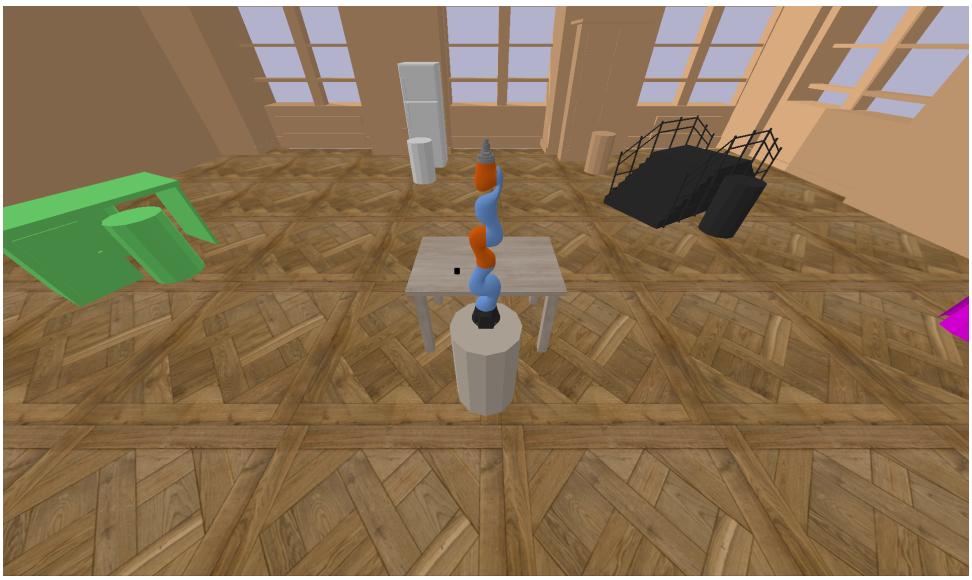


Figure 1.2: Tabledesk workspace with Kuka robot.

Figure 1.3: LBR iiwa robot in real life.

It has many workspaces based on which the environment setup is adjusted. Workspace defines an area around the robot and a list of available tasks. Our primary focus will be concerned with Tabledesk (the figure 1.2) and Collaborative table workspaces. The first is chosen because it is the most simple workspace, which allows training for Reach task, but the other ones with Reach task can also be used (Drawer, Fridge). The second is chosen because it is the only workspace that has a human.

In myGym simulator, we can choose between many robots such as Kuka-IIWA, Franka-Emica, Jaco arm. We will exploit Kuka robot for simplicity reasons, but nevertheless, the other ones also can be used. Kuka is a 7-axis robot arm designed for safe human-robot collaboration in the workspace (the table 1.1). It has joint-torque sensors in all axes to move precisely and detect contact with humans and objects. Kuka, as follows from the definition, has seven degrees of freedom or seven independent parameters that define its state.

In myGym simulator there is a wide choice of tasks. The interaction with an environment during training differs depending on the type of task that is being trained. We will be most interested in Reach task. The main goal of that type of task is to reach an object with a robotic gripper. In other words, the task is to minimize the objective:

Parameter	Value
Width	80 cm
Height	126.6 cm
Length	40 cm
Weight	22 kg
Year	2013
Power	External power supply
Software	Kuka Sunrise.OS

Table 1.1: Parameters of Kuka LBR iiwa robot.

$$\mathrm{d}(o, g) = \sqrt{\sum_{i=1}^{3} (o_i - g_i)^2}$$

where o and g are tuples of Cartesian coordinates of the object and the robotic gripper, respectively. The formula above implies that we are using the Euclidian distance, but it is not mandatory.

To train robots to achieve given tasks reinforcement learning (RL) is used. Reinforcement learning is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning differs from supervised learning in not needing labelled input/output pairs to be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead the focus is on finding a balance between exploration and exploitation.

Chapter 2

Reach gesture

2.1 Human module

To implement Reach gesture task, a human capable of pointing to an object is required. That is why the first thing we need to do is to add a pointing functionality. The already existing myGym module for a robot requires a model to have both a gripper and end effector. Moreover, it is not the best way to represent a human with a robot module. Hence, we introduce a specific human module human.py.

It is very similar to its prototype robot.py with slight differences. For example, the functions for robotic manipulations were simplified:

```
def _run_motors(self, motor_poses):
    self.p.setJointMotorControlArray(
        self.body_id,
        self.motors_indices,
        self.p.POSITION_CONTROL,
        motor_poses,
    )

def _calculate_motor_poses(self, end_effector_pos):
    return self.p.calculateInverseKinematics(
        self.body_id,
        self.end_effector_idx,
        end_effector_pos
    )
```

where the first function is responsible for controlling joints and the second for calculating the parameters of joints by inverse kinematics. By combining these two functions, we can point a human's finger anywhere (the figure 2.1).

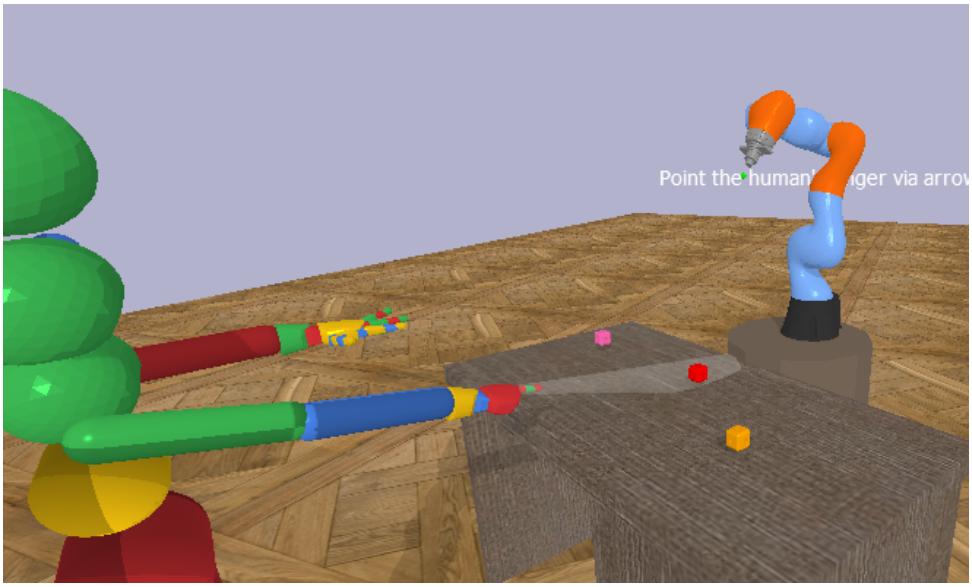


Figure 2.1: Human is pointing at the red cube with his index finger.

The unified robotics description format is an extensible markup language file type that includes the physical description of a robot. It is essentially a 3-D model with information about joints, motors, and mass. To determine the link (finger) for pointing in that URDF, the prefix "endeffector" is used at the end of the link name. Some of the links have also been fixed, making the pointing more realistic (fingers, hand). To indicate a pointing direction a white transparent cone is used as a part of URDF model.

2.2 The nearest object

Reach gesture task requires an ability for determining the object the human is pointing at. To solve this problem, we first need a vector representing a pointing direction. We can get it through the coordinates of the last two links of the index finger. This is where the PyBullet function getLinkState() comes in handy. Getting coordinates of task objects is straightforward thanks to the myGym API - EnvObject.get_position(). When we have all the necessary coordinates, we can start the calculations:

1. Let us define v_1 and v_2 as the points of the last two forefinger joints. Then the vector defining the pointing direction can be computed as

$$\vec{v} = \frac{v_1 - v_2}{\|v_1 - v_2\|}$$

The vector is also normalized to simplify future calculations.

2. Let us define $O \in \mathbb{R}^{n\times 3}$ as the object coordinates matrix. To compute projections , the vector must be in the center. Because projections depend only on the vector of direction, it is enough to center just object coordinates

$$\forall i \in 1, \ldots, n : \; C_i = O_i - v_2$$

where v_2 is the vector beginning and M_i means i-th row of the matrix M. This way, we got a new matrix C with centered coordinates.

3. Further, we can utilize the dot product to compute projections

$$\forall i \in 1, \ldots, n : \; P_i = C_i \cdot \vec{v}\, \vec{v}$$

where $\cdot$ means dot product. We don't need to place $\vec{v}$ norm in the denominator because it is already normalized. This way, we got a new projection matrix P.

4. The next step is computing rejections. This can be done using a definition of rejection

$$\forall i \in 1, \ldots, n : \; R_i = O_i - P_i$$

where R is a matrix of rejections.

5. We will also need a magnitude of those rejections

$$\forall i \in 1, \ldots, n : \; n_i = \|P_i\|$$

where n is a tuple of norms.

6. And finally, the index of the nearest object can be defined as

$$\arg\min_{i \in n} n_i$$

Then we are done. The nearest object is detected. All those steps can be written in a simplified form thanks to NumPy [HMvdW+20]:

```
points -= p2.reshape(1, -1)
# move points relative to the vector's beginning (p2)
scalars = np.dot(points, vector)
# scalar product (as a part of computing projections)
points_proj = scalars.reshape(-1, 1) * vector.reshape(1, -1)
# projections on the vector
points_rej = points - points_proj  # rejections
distances = np.linalg.norm(points_rej, axis=1)
return objects[np.argmin(distances)]
```

2.3 Reach gesture

With all the steps above Reach gesture task can be defined. Reach gesture is Reach task with some minor changes. During training, the human points in a random direction, and depending on the direction, the nearest object is determined using the method from the section 2.2. During testing, it should be possible to select the goal object manually. For that purpose, the function choose_goal_object_by_human_with_keys() was written that utilizes PyBullet API for keyboard interaction. Keypress can be obtained via getKeyboardEvents(). The following code snippet represents the change direction via keys:

```
if self.p.B3G_LEFT_ARROW in key_press.keys()
    and key_press[self.p.B3G_LEFT_ARROW] == 3:
    self.human.point_finger_at(
        move_factor * np.array([0.01, 0, 0]), relative=True)
```

Where the function for pointing is composed of the functions mentioned in the section 2.1. The whole process can be demonstrated by the video. The robot does not reach the object along the best trajectory, but this is because the training was short, and the purpose of the video is to show the choice of the goal object.

In order for the hand to actually move during training, several simulation steps are launched (usually up to 10). This is shown in the video. The text behind the scene is used for hints to the user with PyBullet addUserDebugText().

Chapter 3

Language module

3.1 Auxiliary classes

To build a language module some secondary functions and classes are required. The first auxiliary class, VirtualObject, is a building block of the language module. It wraps the original myGym's EnvObject to make it possible to interact with the object through a natural language. It is called virtual for the purposes of the future expansion of myGym. Precisely, to move an object virtually without a real simulation. There is a natural need for that in multi-step task generation, where the objects should be moved dynamically along with the task generation. Still, the actual simulation, for obvious reasons, is not possible.

The VirtualObject class allows extracting a natural language description from an EnvObject. It converts internal properties like a color or a shape to the corresponding description. There are also the opposite methods, which, given an object list, find the objects most similar to that in the natural language description. Combining these methods one after another, it is possible to uniquely determine the object from a description.

The VirtualEnv class is the core of an internal environment representation. It is made up of the real GymEnv and lists of VirtualEnv objects. Inside it, there are two types of objects, real physical objects and "dummy" target objects (the figure 3.1). The first ones are just EnvObjects, whereas the second ones must always be transparent and not have any collision model. Because target objects serve as a place for the final object destination.

The other important role of the VirtualEnv class is to provide methods for capturing spatial relations. The implemented ones are "left to" and "right to"

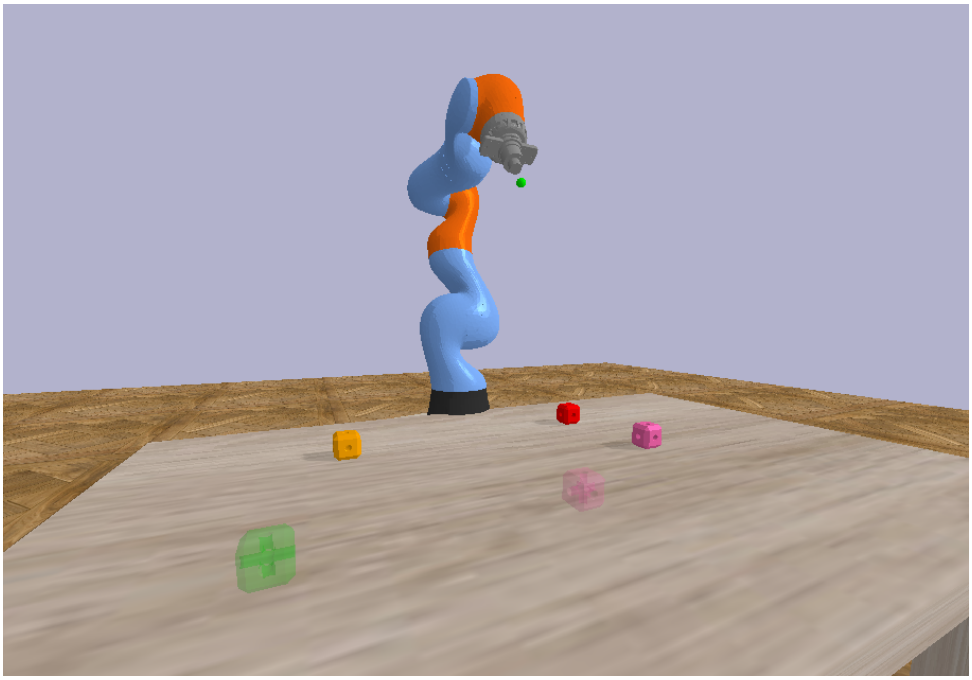


Figure 3.1: Example of a scene with objects of two types. The opaque objects serve the role of real objects, and the transparent one are used to indicate a goal.

but the whole system is built so that it is pretty easy to add new relations. In other words, on the one hand, the class is capable of extracting spatial relations and converting them to the corresponding object descriptions. On the other hand, with the VirtualObject's help, it is capable of determining the unique objects which satisfy given relations. That helps to build a more complex system.

To internally differentiate between initial and target (goal) objects, a new "task_objects" parameter type was proposed as depicted on the figure 3.2. "task_objects" is one of many parameters of the configuration file, which is used to run deterministic training. All configuration files can now accept a new "task_objects" type. Specifically, the one with two lists. With the list of initial objects and the list of goal objects. Initial and goal here mean the same as in the default myGym version. The only difference is that the initial and the goal objects will be picked randomly. That has the purpose of increasing variability and producing more interesting spatial relations.

There was also demand for a new color system. The only available colors now are the ones defined in the corresponding color module. The colors in that module always have a name in a natural language so that the language module can work with strings and not with the RGBA values. That can be done through special methods for conversion in both directions. To add new colors one must append them to the dictionary OPAQUE_COLOR_DICT

```
#Task
"task_type"            :"pnp",
"natural_language_mode" :1,
"task_objects"         :{"init": [
                       {"obj_name": "cube_holes", "fixed": 0, "rand_rot": 0, "sampling_area":[-0.6, -0.1, 0.2, 0.4, 0.075, 0.075]},
                       {"obj_name": "cube_holes", "fixed": 0, "rand_rot": 0, "sampling_area":[-0.3, 0.3, 0.5, 0.6, 0.075, 0.075]},
                       {"obj_name": "cube_holes", "fixed": 0, "rand_rot": 0, "sampling_area":[0.1, 0.6, 0.2, 0.4, 0.075, 0.075]},
                       ],
                       "goal": [
                       {"obj_name": "cube_target", "fixed": 1, "rand_rot": 0, "sampling_area":[-0.2, 0.2, 0.7, 0.8, 0.075, 0.075]},
                       {"obj_name": "cube_target", "fixed": 1, "rand_rot": 0, "sampling_area":[0.3, 0.5, 0.7, 0.8, 0.075, 0.075]},
                       ]},
```

Figure 3.2: New "task_objects" parameter in a configuration file. Instead of a dictionary for every subtask, two lists of any length are used. The first stands for initial objects, and the second for target objects.

in the color module. Some code snippet from it:

```
def get_all_colors(transparent=False, excluding=None):
    excluding = [] if excluding is None else excluding
    return [
        v for k, v in
        (OPAQUE_COLOR_DICT if not transparent
        else TRANSPARENT_COLOR_DICT).items()
        if k not in excluding
    ]

def draw_random_rgba(
        size=None, replace=False,
        transparent=False, excluding=None
    ):
    colors = get_all_colors(transparent, excluding)
    rgba = rng.choice(colors, size=size, replace=replace)
    return tuple(rgba) if size is None
        else [tuple(r) for r in rgba]
```

3.2 Language module

The classes and modules described in the section 3.1 allow to implement a language module. This language module contains templates for every myGym task in a natural language. It can generate detailed object descriptions and combine them into a desired task. For every encoding method, there is also the opposite decoding method for parsing a natural language description. The main idea is quite simple. Any task description has two parts, the description of the task and object clauses. The object clauses have a precise position in a sentence for every task, so there is no problem with an encoding and decoding process. The idea of a language module and its actual representation in myGym simulator was inspired by the LANRO gym [REW22] [RE22] (the figure 3.3).

The NaturalLanguage class also has an API for setting internally available objects. These objects are further used for generating natural language

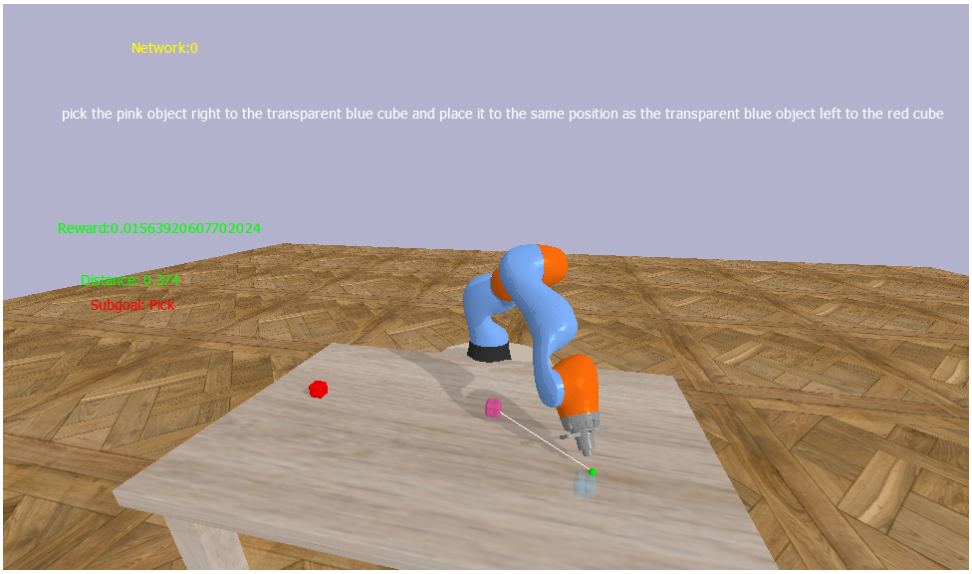


Figure 3.3: Description in natural language of Pick and Place task.

descriptions. The external API is just a function get_venv().set_objects(). It requires passing the corresponding optional argument. The argument can be task_objects for the standard task objects type that myGym uses (i.e., the list of subtasks with "init-goal" pairs). The argument can be init_goal_objects if we want to initialize objects from two lists of different lengths. And finally, the optional argument can be all_objects if we don't want to pass the information about which object is a target and which is only initial.

As can be seen above, it is not really necessarily to understand the underlying structure in the form of auxiliary classes. All the basic API is concentrated in the NaturalLanguage class. One of the first exciting functions is generate_subtask_with_random_description(). Before calling this function, a user must set some objects through the API above. Then it generates a task based on the environment task type and previously provided objects. The natural language description of the task is saved internally, but the information about the task (e.g., objects, a task type) is lost for the purpose. The final description can be obtained through the get_previously_generated_subtask_description() function.

The information about the generated task is lost because it is assumed to extract the information using extract_subtask_info_from_description(). The function gets a description and extracts all information related to the task, i.e. initial and goal objects, a task type, a reward type, number of neural networks. Clauses in the current natural language module have fixed positions, so extracting is straightforward. Generally, a task type can be extracted from the first words, and then the whole sentence can be divided into parts containing only object clauses, simplifying further parsing.

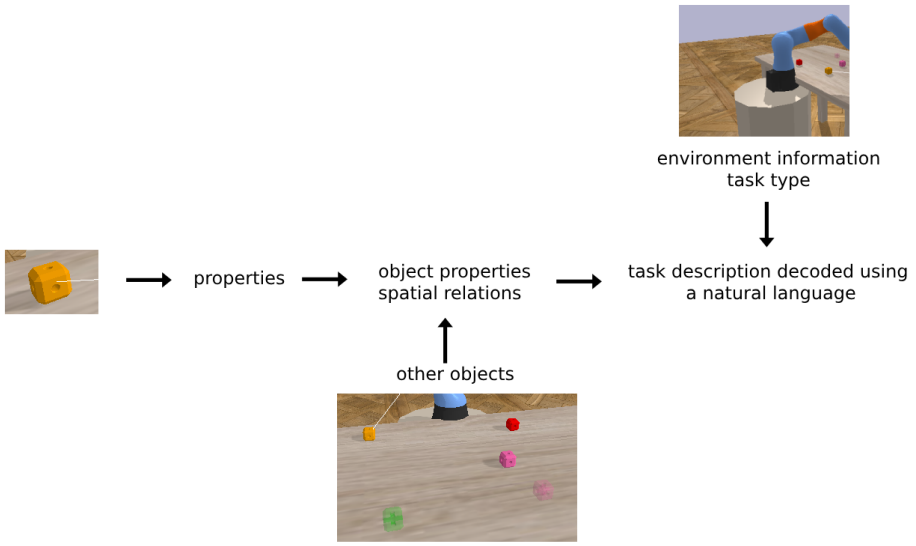


Figure 3.4: The process of generating a natural language description in myGym simulator.

The other useful function is generate_random_description_for_current_subtask(). The function is used to produce a description of an already existing subtask. It can be seen behind a scene in white font. Otherwise, the function can be used to produce a description of any subtask which is set to the current in the actual environment.

There is also deprecated functionality, the code which was written during the project, but suddenly turned out to be not suitable and not supported at that moment. The code can still be used as a template in the future. This code is repesented by the functions generate_new_tasks(), _generate_new_subtasks(), and _generate_new_subtasks_from_1_env(). The latter is the core function; it gets a tuple of a string (a description) and a virtual environment (a VirtualEnv instance) and returns a list of such tuples. Basically, it generates a new environment from the given one and extends the description with the new subtask. The environment is different because the newly generated subtask is already completed in that environment. This basic idea can be used further to generate many parallel environments with different subtasks. This is done by the two remaining functions. generate_new_tasks() adds new functionality in the sense that it allows restricting of the width (the maximum number of environments) and the depth (the number of subtasks in one environment).

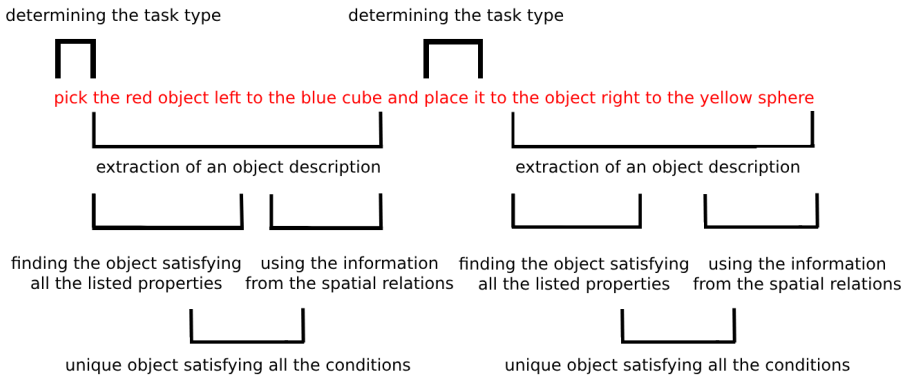


Figure 3.5: Example of parsing a natural language description. The sentence is divided into parts containing the information about a task type and objects clauses. The object is uniquely determined by the combination of properties and spatial information.

Chapter 4

Results

4.1 Evaluation

Reach gesture task was trained on 10000 steps. The number of steps is so low because Collabtable workspace is inherently slow. It is approximately 150 times slower than Reach task. However, the decreasing loss and value loss in the figure 4.1 shows that the robot is capable of learning using human gesture. The value function loss is the mean loss of the value function update. It correlates to how well the model is able to predict the value of each state.

NL Reach task was trained on 1 million steps. The results are shown in the video. Kuka is capable of reaching objects based solely on the description. For the purposes of fast training, the number of objects was limited, but otherwise there can be much more objects and more complex descriptions taking into account spatial relations can be used.

NL Pick and Place task was tested in 5 scenarios (the figure 4.2). Using different robots, different control mechanisms, and main reinforcement learning algorithms. One can control the robot's links by setting the absolute or relative position of the gripper or directly by choosing the angles of the joints. But the most important thing is that in all proposed scenarios, there is a positive trend - the percentage of successful episodes is growing with time. Thus, it can be stated that robot learns to reach objects using verbal description and human gesture.

Finally, the three main tasks were trained with more steps (the figure 4.3). NL Reach and NL Pick and Place with more than 800000 steps, and NL Reach Gesture with 400000 steps. The main trend is not perfect, but nevertheless, the final value is better than the initial value everywhere. NL

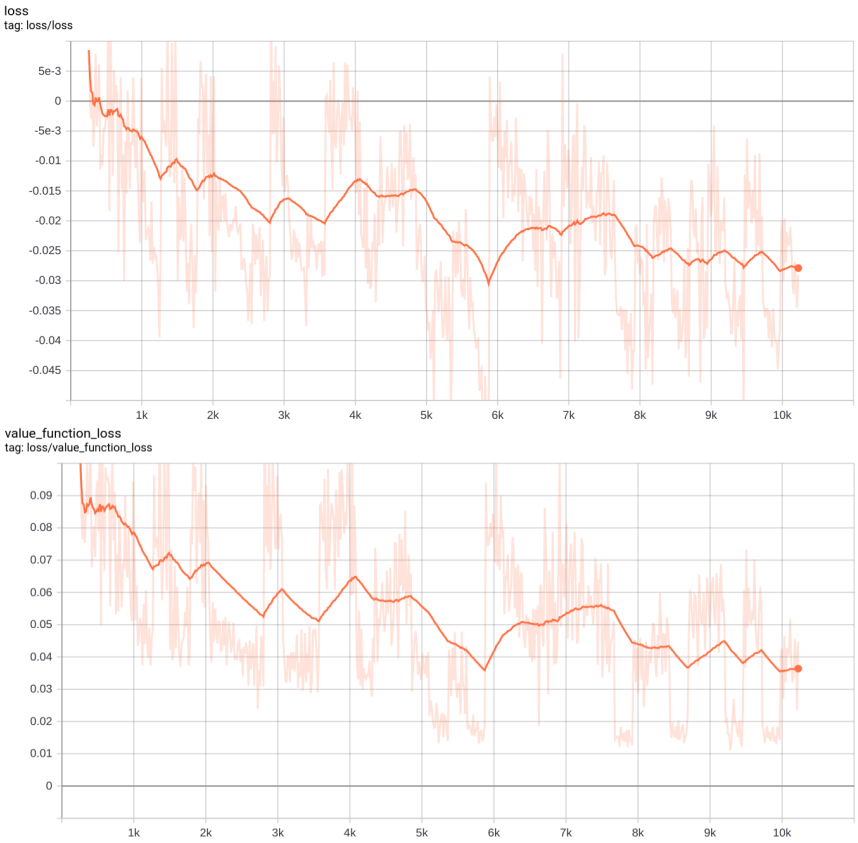


Figure 4.1: Loss and value function loss of Reach gesture task from the tensorboard.

Pick and Place has the best mean reward curve, whereas NL Reach gesture has the worst. This is due to the slow nature of Collabtable workspace and fewer steps overall.

4.2 Conclusion

In this article, we have shown the implementation of basic robot control in myGym simulator through natural language and human gestures. Sentences in natural language are not overly complex; however, the language module is written in such a way that it can be easily extended to further relations and tasks. Reach gesture task can be further extended. For example, the object a person is pointing at will only be determined based on the camera image. Most importantly, the foundation was laid for human interaction through gestures and language.

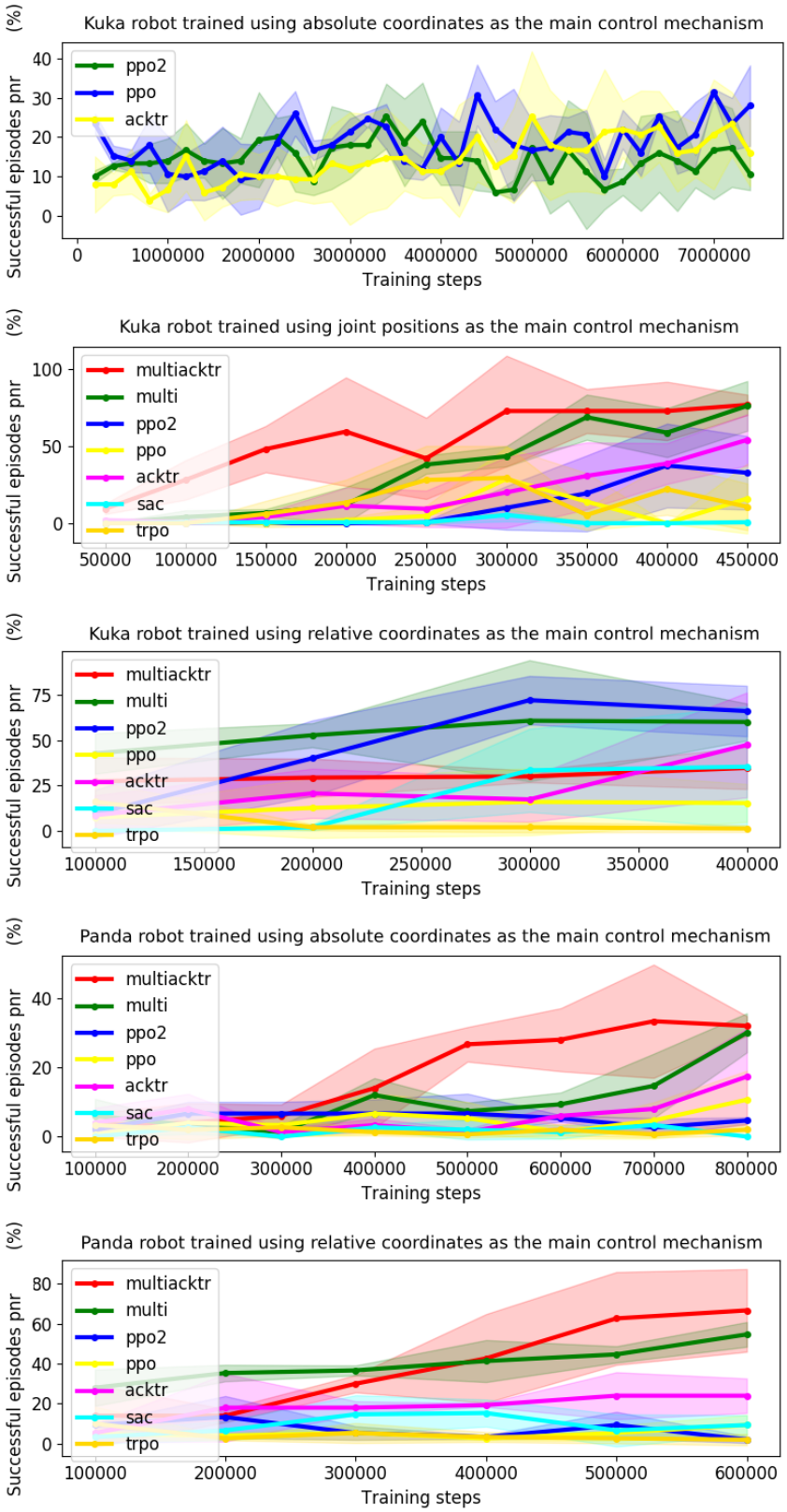


Figure 4.2: Successful episodes ratio of NL Pick and Place task in different scenarios.

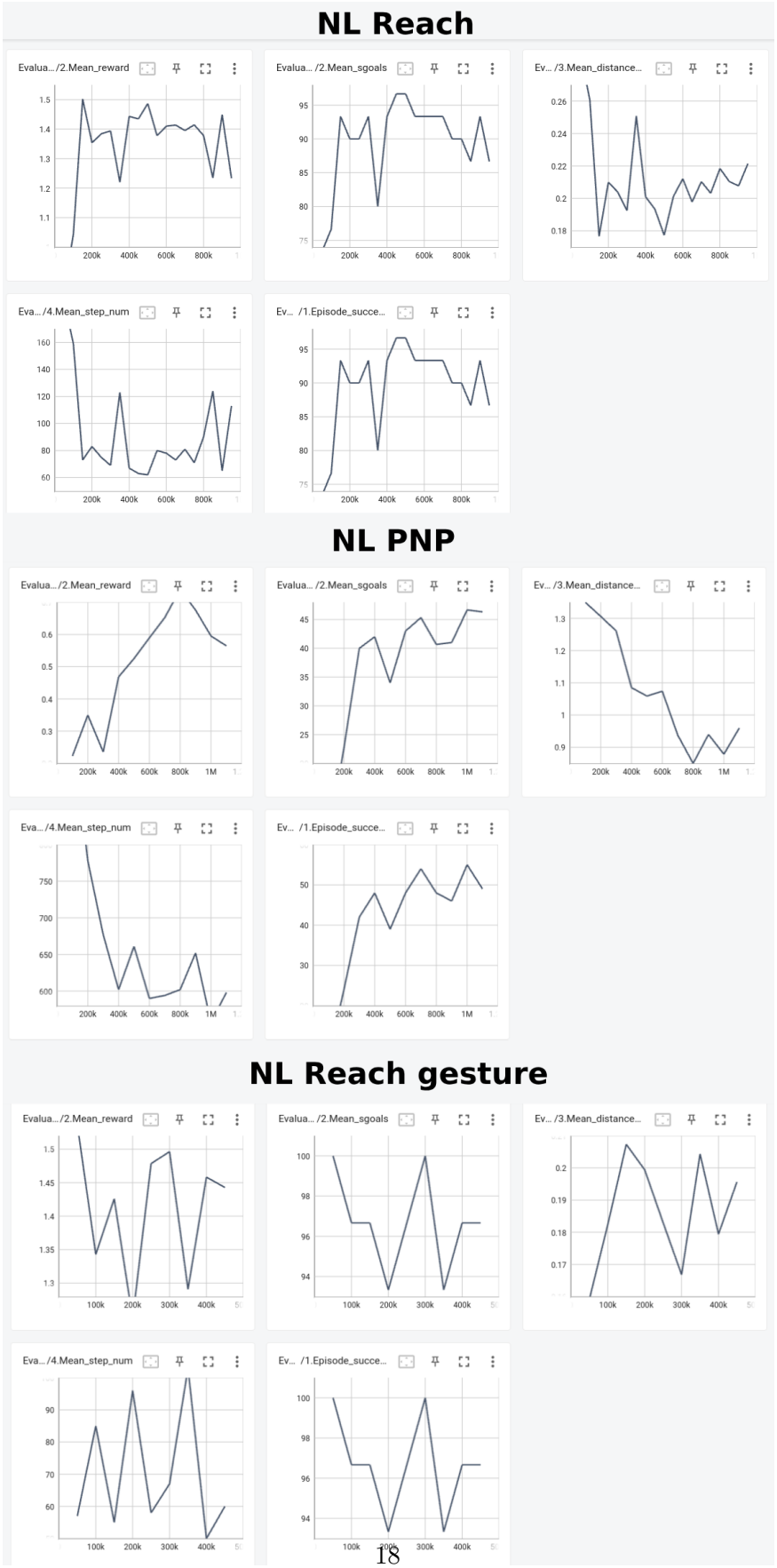


Figure 4.3: Evaluation of NL Reach, NL Pick and Place and NL Reach gesture for a longer number of steps.

Bibliography

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant, *Array programming with NumPy*, Nature **585** (2020), no. 7825, 357–362.

[RE22] Frank Röder and Manfred Eppe, *Language-conditioned reinforcement learning to solve misunderstandings with action corrections*, 2022.

[REW22] Frank Röder, Manfred Eppe, and Stefan Wermter, *Grounding hindsight instructions in multi-goal reinforcement learning for robotics*, 2022.

[RSKP10] Jagdish Lal Raheja, Radhey Shyam, Umesh Kumar, and P. Bhanu Prasad, *Real-time robotic hand control using hand gestures*, 2010 Second International Conference on Machine Learning and Computing, 2010, pp. 12–16.

[VSM+20] Michal Vavrecka, Nikita Sokovnin, Megi Mejdrechova, Gabriela Sejnova, and Marek Otahal, *mygym: Modular toolkit for visuomotor robotic tasks*, 2020.