ETH zürich

Soft Robotics Laboratory

Semester Thesis

Learning a Policy for a Biomimetic Tendon-Driven Hand Combined with a Robotic Arm

Sebastiano Oliani

11 January 2024

Supervision

Yasunori Toshimitsu
Prof. Dr. Robert Katzschmann

Abstract

Dexterous Manipulation is an area of robotics in which multiple manipulators, or fingers, cooperate to grasp and manipulate objects. Biomimetic, dexterous robotic hands have the potential to replicate much of the tasks that a human can do, and to achieve status as a general manipulation platform.

Thanks to the recent advances in reinforcement learning (RL) combined with GPU-based highly parallelized simulations capable of simulating thousands of robots in parallel, RL-based controllers have become more scalable and approachable. Although simple tasks in industrial environments have already been solved by developing simple grippers, more complex tasks cannot be achieved without dexterous end-effectors. At the Soft Robotics Lab, the biomimetic tendon-driven Faive Hand has been developed to exploit reinforcement learning simulation environments and bring robotic hands into the real world.

This work aims to enhance the current applications of the Faive hand by mounting it on a Franka Emika Panda arm, and using it to stack cubes on top of each other. Both the Faive Hand and the robotic arm are trained with reinforcement learning using the Nvidia Isaac Gym simulator. Thus, control policies are optimized by creating a simulation of the task and collecting data according to an episodic setting.

After the reinforcement learning training reaches the desired behaviour, the control policies are implemented in a real physical environment. Objects in the scene are detected thanks to a perception system. As may happen when testing policies that have been trained using an episodic setting, many issues are encountered in closing the sim-to-real gap. Even if some individual components of the entire control pipeline are separately working, the task is not fully achieved in the end.

Summary of notation

$\boldsymbol{q}_{arm}$	Generalized joint arm coordinates
$\boldsymbol{q}_{hand}$	Generalized joint hand coordinates
$\boldsymbol{q}_{max}$	Generalized coordinates upper limit
$\boldsymbol{q}_{min}$	Generalized coordinates lower limit
$\boldsymbol{q}_{start}$	Generalized coordinates initial condition
$\boldsymbol{p}_{grab}$	Position of the grabbing cube
$\boldsymbol{p}_{ref}$	Position of the reference cube
$\boldsymbol{p}_{hand}$	Position of the hand's palm
$\boldsymbol{p}_{fingertip}$	Position of the hand's fingertips
$\boldsymbol{p}_{x}$	Position X-coordinate
$\boldsymbol{p}_{y}$	Position Y-coordinate
$\boldsymbol{p}_{z}$	Position Z-coordinate
$\hat{\boldsymbol{j}}$	Y-coordinate unit vector
$\hat{\boldsymbol{k}}$	Z-coordinate unit vector
$\boldsymbol{l}_{grab}$	Length of the grabbing cube
$\boldsymbol{a}$	Actions
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2
$\mathcal{U}(a, b)$	Uniform distribution within a and b

Contents

1 Introduction

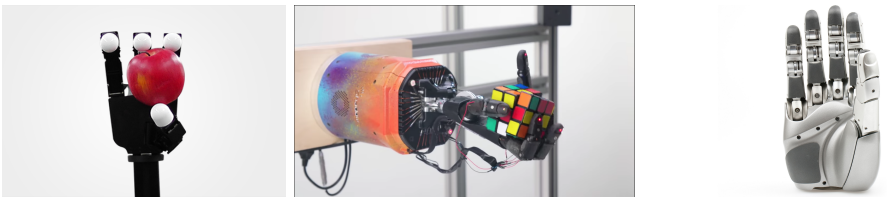
1.1 Motivation

In industry, robotic assembly methods may achieve high precision, accuracy, and reliability. However, these methods can be highly restrictive. They often use expensive equipment, require custom fixtures, have high setup times (e.g., tooling design, waypoint definition, parameter tuning) and cycle times, and are sensitive to variation (e.g., part appearance, location). Custom tooling and part-specific engineering are also cost-prohibitive for high-mix, low-volume settings. In the last decades, physics simulation has become a powerful tool for robotics development. Simulators have primarily been used to verify and validate robot designs and algorithms. Recent research has demonstrated a host of other applications: creating training environments for virtual robots, generating large-scale grasping datasets, inferring real-world material parameters, simulating tactile sensors, and training RL agents for manipulation and locomotion. Compelling works have now shown that RL policies trained in simulation can be transferred to the real world [1].

Dexterous manipulation is one of the hottest topics of research these days. Dexterous manipulation, requiring precise control of forces and motions, cannot be accomplished with a conventional robotic gripper; fingers or specialized robotic hands must be used [2]. Although some simple tasks in the industrial environment have been solved, the need for robots that can help humans in some unstructured environments such as the domestic environment (e.g., helping blind people with daily routines) and some dangerous environments (e.g., nuclear decommissioning) is significant. When facing a complex unstructured environment, one robot needs to deal with a lot of tasks and, ideally, one robot needs to carry different end-effectors for different tasks, which is unpractical. Hence, the ability to operate with the dexterity of the robot is necessary [3].

1.2 State-of-the-Art

Over the past decades, many dexterous multi-fingered hands have been designed. In 1984, the Center for Engineering Design at the University of Utah, and the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology designed the UTAH/MIT hand with three fingers and a thumb aiming at machine dexterity [4], and later HIT developed the DLR/HIT Hand II [5]. Also, there are some commercial products such as the Shadow Hand [6] and the Allegro hand[1] (see Figure 1). Apart from the dexterous humanoid robotic hands, some simpler robotic manipulators with fewer fingers and lower degrees of freedom (DoF) have been designed for better robustness and lower price. The current applications of robotic hands in the factories still use traditional engineering and analysis techniques. Typically, some robots with simple end-effectors are widely used in the manufacturing industry for packaging and palletizing. Similarly, agricultural robotic hands with several end-effectors and painting robotics are good examples of the application of robotic hands in the structured industry environment [3].



(a) (b) (c)

Figure 1: (a) Allegro Hand (b) Shadow Hand (c) DLR Hand II

Although the mechanical design of smart manipulators has improved greatly, the actual dexterity of the robotic hands is far inferior to that of the human hand. On one hand, lots of sensors and actuators of the human hand make it almost impossible to design a robotic hand which is similar to the human hand, and on the other hand, the control of the robotic hand to realize dexterous

[1] Website: http://wiki.wonikrobotics.com/AllegroHandWiki/index.php/Allegro_Hand_v4.0

manipulation is still an urgent problem to solve. Model-based approaches fail in real applications due to inaccuracies in the models, and because algorithms must change as the object or the manipulator changes. In contrast, reinforcement learning is more suitable for dealing with the sequential decision problem. Therefore, the combination of deep learning and reinforcement learning called deep reinforcement learning is proposed to realize more complicated problems involving perception and decision-making. However, the application of deep reinforcement learning to dexterous manipulation has some disadvantages. First, the sparse reward makes the training hard, and for complex tasks, it is time-consuming and the requirement of computing power is high. Furthermore, deep reinforcement learning requires many samples obtained by trial and error, which are nearly unavailable in a robotic system. To solve this problem, besides the improvement of the reinforcement learning algorithm, two solutions are usually considered: learning from demonstration and transferring the policy learned in simulation to reality. These two approaches will greatly enhance the efficiency of the algorithm.

Another key element when working with reinforcement learning is the sim-to-real gap. This term describes the discrepancies between simulation and real robots that make the transformation challenging. Transforming the policy directly to the real world may cause various consequences, the lesser of which is a decline in success and the more serious of which is the instability of the system that may destroy the robotic hands or the environment. Hence, closing the sim-to-real gap is the main issue when mentioning the sim-to-real problem. The sim-to-real problem is not unique to the field of reinforcement learning or dexterous manipulation, but a general problem in machine learning for robotics. The main approach widely used for closing the sim-to-real gap is domain randomization. The main idea of domain randomization is to randomize the simulation with disturbance, i.e. variations in the values of the main parameters of the simulation like friction coefficients, mass, contact model, etc. Through exposure to various environments, the agent trained in simulation can adapt to a wide range of environments, and, in the end, succeed in closing the sim-to-real gap.

The implementation of reinforcement learning algorithms using robotic arms combined with human-like hands is a very active research topic. In those cases, imitation learning is adopted to improve the training and transfer the learned policy into the real environment. If imitation learning is not implemented, then mostly standard-grippers are used [7]. Common tasks covered in nowadays research are opening a drawer, opening a door, picking up a ball, in-hand rotation, etc.

1.3 Current set-up developed at Soft Robotics Lab

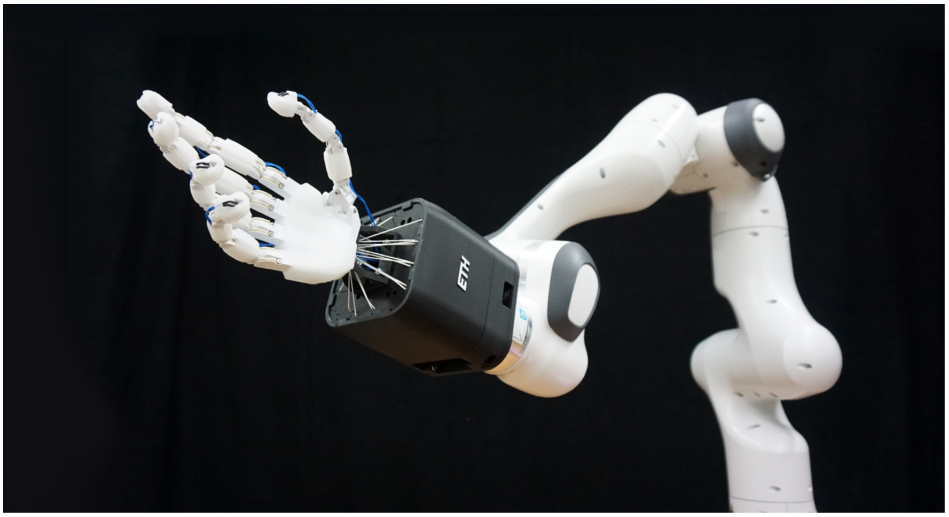


Figure 2: Franka Emika and Faive Hand Proto0.

The Faive Hand has been developed at the Soft Robotics Lab. It is a biomimetic dexterous tendon-driven robotic platform for exploring dexterous manipulation. The so-called Proto0-version of the

hand uses 3D-printed components and servo motors for accessible and simple manufacturing. However, in addition to the challenges inherent to controlling a high-DoF robotic hand for manipulation, this hand has features that do not exist in other dexterous hands trained with reinforcement learning, such as rolling contact joints that rotate without a fixed axis of rotation. The hand currently does not have internal joint angle encoders, but the joint angles must be estimated from the tendon length, which can be calculated from the servo motor angles.

The Faive Hand contains 11 actuatable degrees of freedom, three in the thumb and two for each of the other fingers. Each finger contains a coupled joint at the distal end, and thus there are 16 joints in total. All of the joints are implemented as rolling contact joints, except for two hinge joints mounted in the thumb. These rolling joints are composed of two articulating bodies with adjacent curved contact surfaces connected by a pair of crosswise ligament strings. They have advantages such as impact compliance, low friction or greater range of motion [8].

These features were implemented in the simulation framework. Then the Faive Hand was simulated in Nvidia Isaac Gym to run a closed-loop RL-trained policy (see Figure 3). The first task that was chosen with the given set-up was the in-hand rotation of a sphere in a target direction.

The policy is trained with reinforcement learning with advantage actor-critic (A2C) using asymmetric observations (where different sets of observations are given to the actor and critic). The PPO algorithm [9] with the implementation from the open-source repository *rl games*[2] is adopted. MLP networks are used as actor and critic.

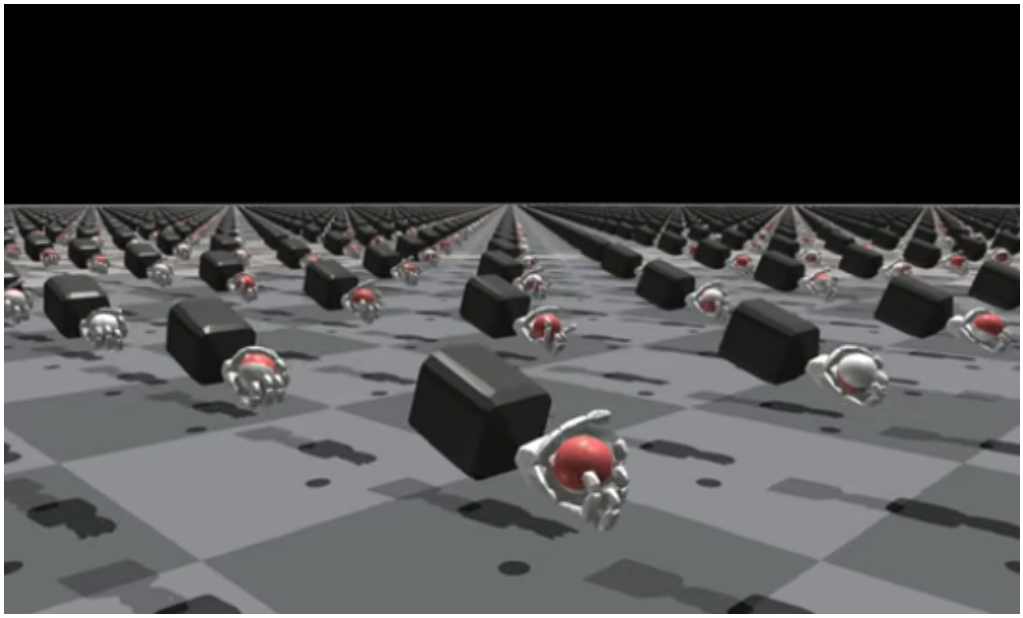


Figure 3: Faive Hand training in Isaac Gym.

As part of the set-up, the Faive Hand is mounted on a Franka Emika Panda arm[3]. This industrial robot arm has seven degrees of freedom, 18 kg weight and a 3 kg payload. It originally mounts a two-finger gripper, but, to develop the Semester Thesis, the original gripper will be substituted by the Faive Hand.

[2]D. Makoviichuk and V. Makoviychuk, "rl-games: A high-performance framework for reinforcement learning," https://github.com/Denys88/rlgames, May 2021.
[3]Website: https://www.franka.de/ (see Figure), and datasheet: https://www.phytronrobotics.com/files/136589731.pdf

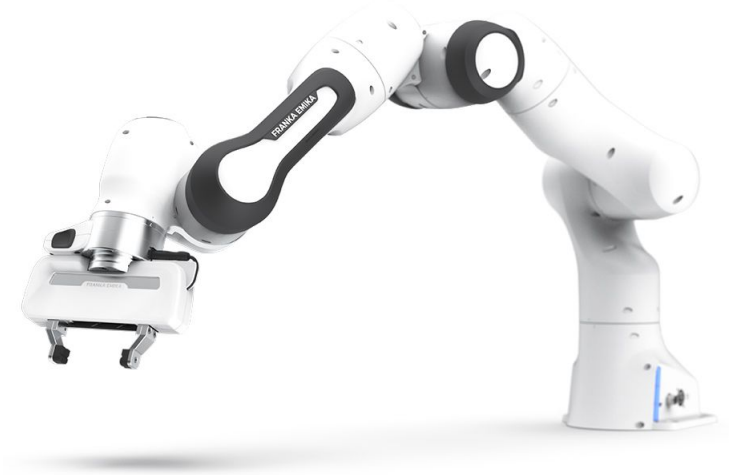


Figure 4: Franka Emika Panda with its standard two-finger gripper.

This work uses a new version of the Faive Hand, called Proto2-version (see Figure 5). This newer version uses rolling contact joints only, even for the thumb. It gains the ability to perform abduction movements with the finger and it revises the thumb anatomy to match the human hand better. This new version consists of 16 actuatable degrees of freedom, four in the thumb and three in the other fingers.

Figure 5: Faive Hand Proto2.

2 Background

As robotics hardware becomes more complex and capable, the importance of simulation tools increases. Simulators play a key role in training robots, improving the learning process's safety and iteration speed through hardware accelerators. Training inside simulators offers an efficient and scalable platform via trial and error with no safety issues observed in the real world. MuJoCo[4], owned by Google DeepMind, and Isaac Gym[5], owned by Nvidia, are exploited to develop this Semester's Thesis.

2.1 MuJoCo

The name MuJoCo stands for Multi-Joint dynamics with Contact [10]. It represents the state in joint coordinates and simulates contacts using the Linear Complementary Problem (LCP). It has several unique features which are rarely needed for simulation purposes but greatly facilitate control applications:

- Models are created in XML format and then compiled automatically into low-level data structures optimized for runtime computation.
- The same dynamical system can be evaluated in parallel for different states and controls. This is useful for approximating derivatives via finite differencing, which enables numerical optimization.
- Inverse dynamics can always be computed even in the presence of contacts and equality constraints.
- Actuator dynamics such as the pressures inside pneumatic or hydraulic cylinders as well as the activations of biological muscles can be modeled. Actuators can transmit forces via linkages or tendons too.

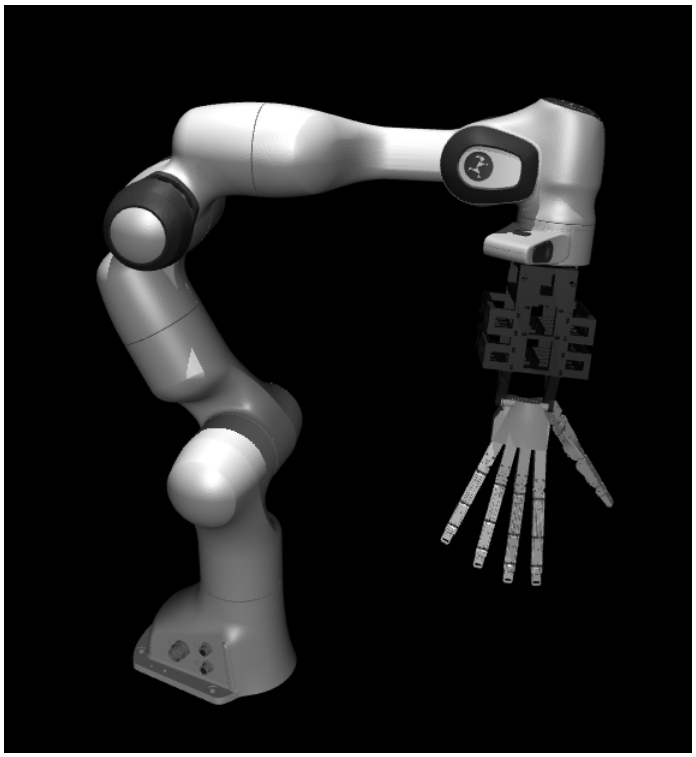


Figure 6: Franka Emika and Faive Hand Proto2 visualized in Mujoco.

[4] https://mujoco.org/
[5] https://developer.nvidia.com/isaac-gym

2.2 Isaac Gym

Isaac Gym allows training policies for a wide variety of robotics tasks using GPU (see Figure 7). Both physics simulation and the neural network policy training reside on GPU and PyTorch tensors[6]. This leads to fast training times for complex robotics tasks on a single GPU with 2-3 orders of magnitude improvements compared to conventional reinforcement learning training that uses a CPU-based simulator and GPU for neural networks [11].

Through Isaac Gym, it is possible to create and populate a scene with robots and objects, supporting loading data from the common URDF and MJCF file formats. Each environment can be duplicated many times in different instances, and Isaac Gym allows varying parameters between them, e.g. via Domain Randomization.

Environments are simulated simultaneously in parallel without interaction with other environments. Isaac Gym also includes a basic Proximal Policy Optimization (PPO) implementation and a straightforward reinforcement learning task system.

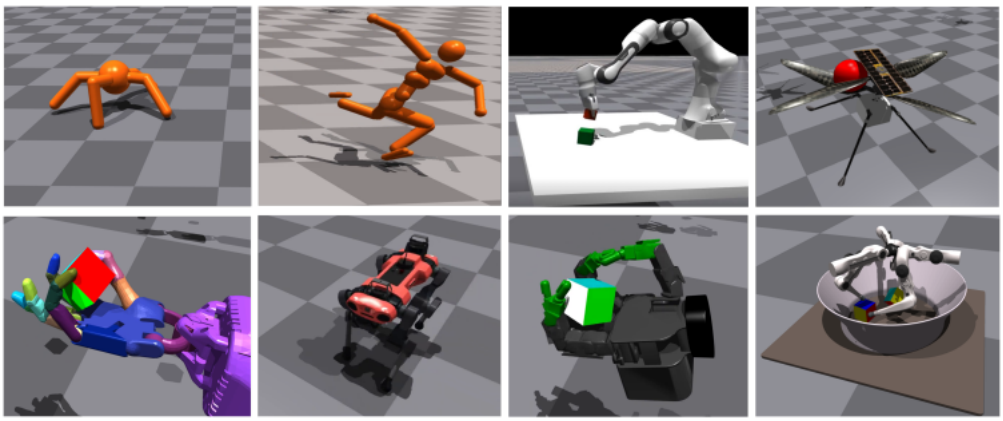


Figure 7: Isaac Gym allows high-performance training in a variety of robotics environments. Top: Ant, Humanoid, Franka-cube-stack, Ingenuity. Bottom: Shadow Hand, ANYmal, Allegro, TriFinger.

[6] https://pytorch.org/

3 Methodology

This Semester's Thesis has been pursued from the beginning of September 2023 to the beginning of January 2024. This section describes the task to implement, the approach, the results of the simulations, the implementation set-up and the final experiment results.

3.1 Task selection

In [8], the Faive Hand Proto-0 has been trained in Nvidia Isaac Gym to rotate a ball along the y-axis. To enhance the possible applications, the following Semester Thesis wants to implement a reinforcement learning algorithm to stack cubes on top of each other in a real environment using the Faive Hand as the end-effector of the robotic arm. The entire system composed of Franka Emika Panda and Faive Hand is trained in simulation to obtain a Reinforcement Learning policy. The main goal is to obtain a working policy by only using the results coming from the simulation. Therefore, more advanced methods as imitation learning that exploits human demonstrations are not used. A similar task is already available as an example in Nvidia IsaacGymEnvs (see Figure 8) and its implementation can be found on GitHub[7]. Its rewards, observations and control law were considered as an initial reference in the implementation of this Semester's Thesis.

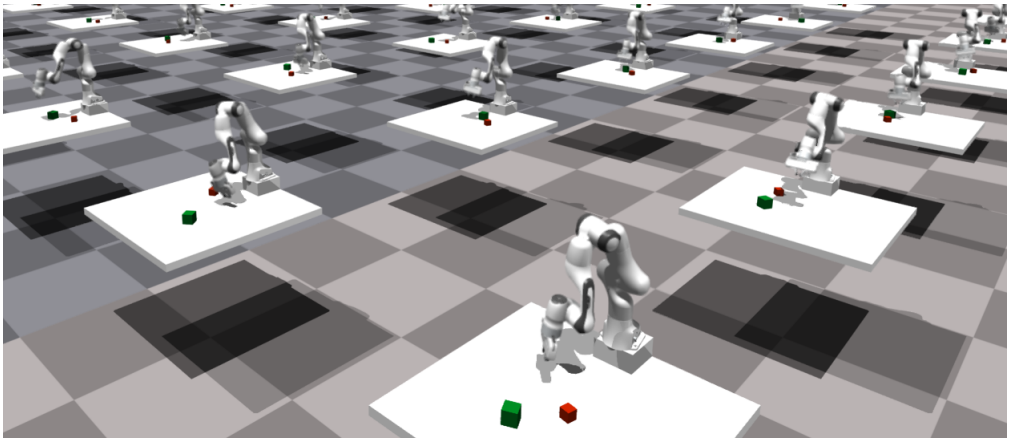


Figure 8: Franka-Cube-Stack implementation from Nvidia IsaacGymEnvs.

3.2 Approach

3.2.1 Simulation environment

First of all, a 3D model of the robotic arm combined with the Faive Hand is created using MuJoCo (see Figure 6). In particular, the 3D model of the robotic arm is open-source, and can be found in the MuJoCo Menagerie, a collection of high-quality models curated by Google DeepMind and published on GitHub. Then, similarly to what was done in [8], the GPU-based simulator Nvidia Isaac Gym is used to simulate the robot for training the policy. 4096 environments are simulated in parallel on a single NVIDIA A10G GPU. The simulation is run at 60 Hz, while the policy runs every three steps, resulting in a 20 Hz policy. The data are collected using Weights & Biases[8] while running the training in Nvidia Isaac Gym (see Figure 9). In this way, it is possible to collect short videos of the agent's behaviour, and plot quantities of interests, like rewards or observations, that can be further exploited to evaluate the results of the training. Crucially, to close the sim-to-real gap, the different elements of the simulation environment, like table dimensions and reference frames, are matched to coincide with the one found in the real world.

[7] https://github.com/NVIDIA-Omniverse/IsaacGymEnvs/blob/main/isaacgymenvs/tasks/franka_cube_stack.py
[8] https://wandb.ai/site

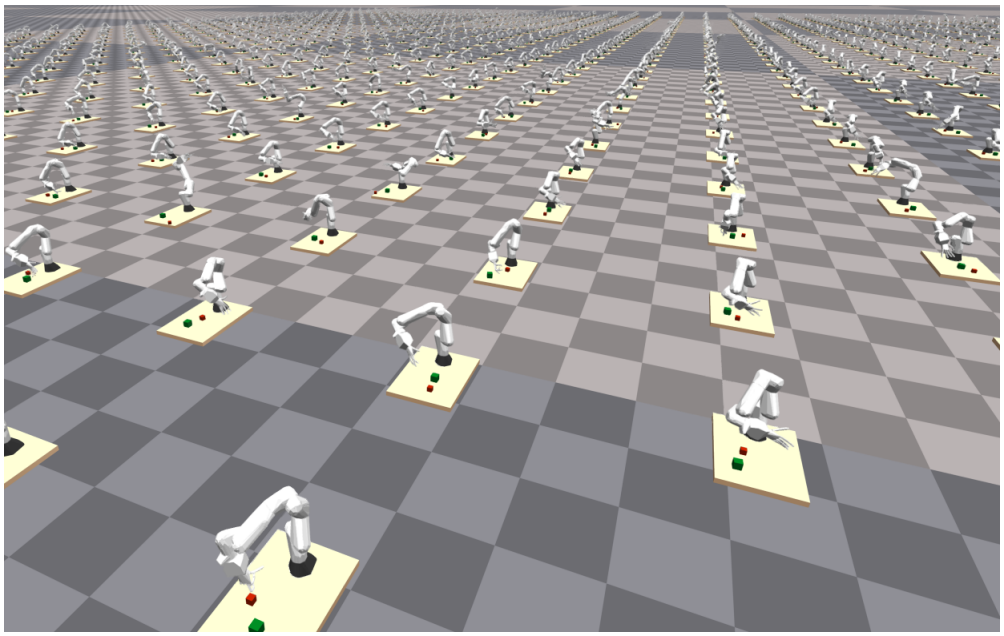


Figure 9: Faive Hand Proto2-version combined with Franka Emika Panda visualized inside the training environment in Isaac Gym.

3.2.2 Rewards and observation space

The rewards and penalties reported in *Table 1* are used to train the system in Isaac Gym. All of them are designed such that they are continuous and output values between -1 and 1. Then, they are scaled by the respective weight. Only the reward corresponding to the correct stack of the cube is expressed as a boolean value. While the purpose of *Action penalty*, *Reference joint penalty*, *Joint limits penalty*, and *Drop penalty* is to increase the safety of the robot movements, the remaining rewards are designed in such a way that they drive the learning process of the robot. First, the robot minimizes the distance of the hand from the grabbing cube. Second, it lifts the cube while maintaining a proper palm orientation. Then, it minimizes the distance between the two cubes, and finally, it stacks them (see Figure 11). In particular, the success condition, namely the correct stack of the cube, is achieved when three conditions are satisfied:

1. the distance of the cubes' centers in the x-y plane is lower than a given threshold.
2. the distance of the cubes along the z-axis is lower than a given threshold (see in more detail in Figure 10).
3. the distance of the hand from the grabbed cube center is bigger than a given threshold.

Reward	***Formula***	***Weight***	***Justification***
Action penalty	$\|\|\boldsymbol{a}\|\|_2$	-0.001	Prevent large actions
Reference joint penalty	$\|\|\boldsymbol{q}_{arm} - \boldsymbol{q}_{start}\|\|_2$	-0.1	Prevent arm joints from moving from the reference position
Joint limits penalty	$\|\| \max(\mathbf{0}, -\boldsymbol{q}_{min} - \boldsymbol{q}_{arm}, \boldsymbol{q}_{arm} - \boldsymbol{q}_{max}\|\|_2$	-0.1	Prevent arm joints to move beyond the joint limits
Drop penalty	$\|\|\boldsymbol{p}_{grab} - \boldsymbol{p}_{ref}\|\|_2 > 50\,\text{cm}$	-1.0	Prevent cube to be dropped
Hand-cube distance	$1 - \tanh(10\frac{\|\|\boldsymbol{p}_{grab} - \boldsymbol{p}_{hand}\|\|_2 + \sum_{n=1}^{5} \|\|\boldsymbol{p}_{fingertip_i} - \boldsymbol{p}_{grab}\|\|_2}{6})$	0.1	Reward distance from the hand to the cube
Cube lifting	$clip(0, \frac{\boldsymbol{p}_{z_{grab}} - \boldsymbol{l}_{grab}}{0.04}, 1)$	1.5	Reward the lift of the cube
Palm orientation	$\langle \hat{\boldsymbol{j}}_{palm}, \hat{\boldsymbol{k}}_{world} \rangle$	0.01	Reward the palm for facing the table
Cube-cube distance	$\|\|\boldsymbol{p}_{grab} - \boldsymbol{p}_{ref}\|\|_2$	2.0	Reward distance from the cubes
Correct stack	$\|\|\boldsymbol{p}_{xy_{grab}} - \boldsymbol{p}_{xy_{ref}}\|\|_2 < 2\,\text{cm} \wedge \|\|\boldsymbol{p}_{z_{grab}} - \boldsymbol{p}_{z_{ref}}\|\|_1 < 2\,\text{cm} \wedge \|\|\boldsymbol{p}_{hand} - \boldsymbol{p}_{grab}\|\|_2 > 8\,\text{cm}$	400.0	Reward the correct stack of the cube

Table 1: Rewards and penalties used during training.

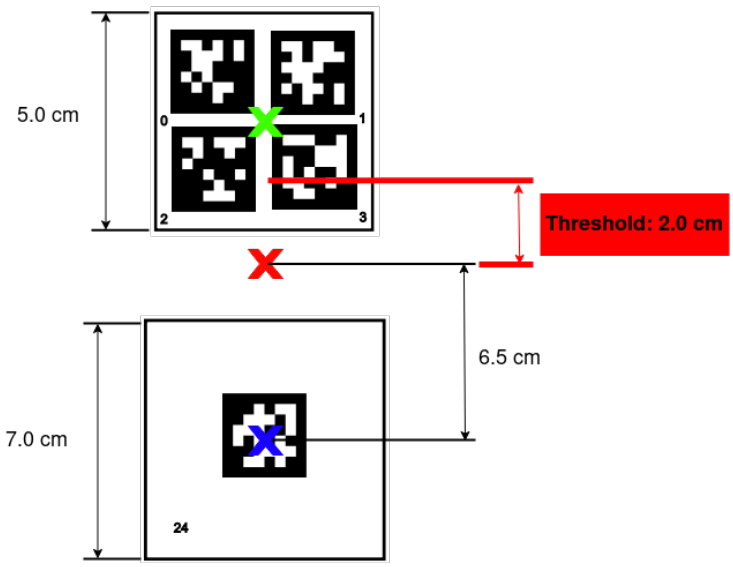


Figure 10: Condition to stack the grabbing cube on the reference cube. The center of the grabbing cube (green) must be inside the red threshold.

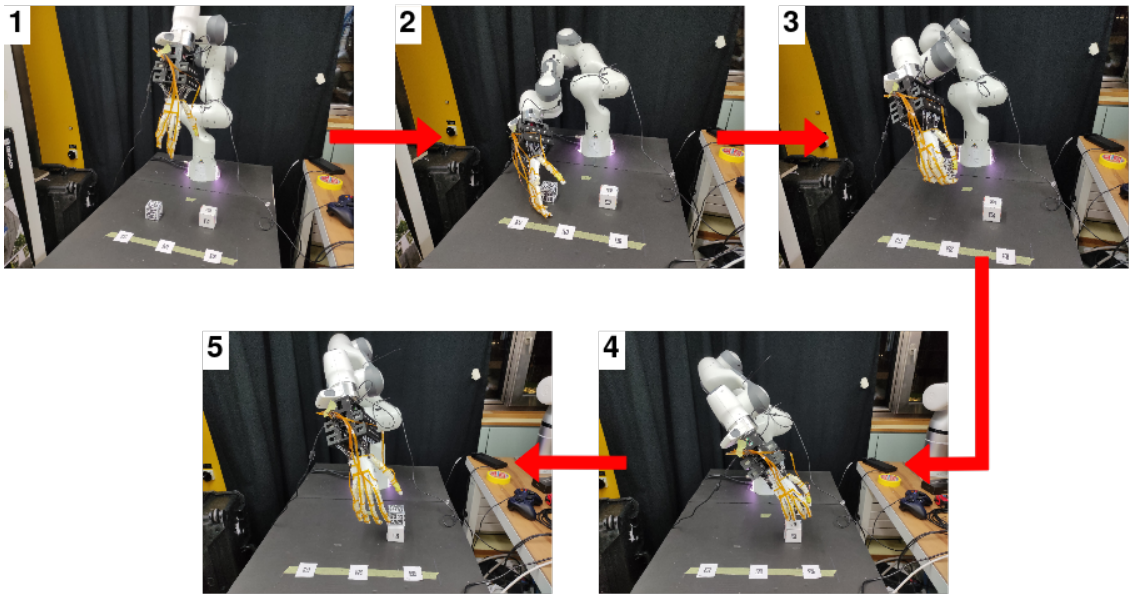


Figure 11: Sequence of pictures representing the expected behaviour of the robot. The robot has been manually moved to take the pictures.

The same observations used by the actor and the critic networks are reported in *Table 2*. The agent uses both proprioceptive and exteroceptive data, due to the need to know objects' position and orientation. As part of the implementation set-up, two cameras are used to estimate the pose of the cubes in the environment, and then generate proper commands for the robot to interact with the system. Therefore, the cubes are covered with markers to be detected by the vision system. It is important to notice that no measurements coming directly from the hand are used. This is useful because it is not possible to measure directly the hand joint angles, which are expressed as a non-linear mapping with the tendon length, and affected by noise due to slack in the tendons. Furthermore, Isaac Gym ignores the tendons-level information and considers the hand only as a joint-level robot. The hand joint angles used by dof_pos_target and *actions* are the output of the actor network in the simulation, and not a measurement coming directly from the hand, for example of the angles read by the motor encoders.

Observations	***Dimension***	***Description***
obj_pos	3	Cube position
obj_quat	4	Cube quaternion
eef_pos	3	End-effector position
eef_quat	4	End-effector quaternion
hand_to_obj_pos	3	Hand-cube relative position
obj_to_goal_pos	3	Cubes relative position
dof_position	7	Arm joint angles
dof_pos_target	23	Hand and arm joint angles
actions	23	Hand and arm joint commands

Table 2: Observations used during training for the actor and critic networks in Isaac Gym.

3.2.3 Action space

Joint position control is adopted to command both the robotic arm and the biomimetic hand. The action $\mathbf{a}$ expresses the relative change in the joint angle command. It is first clipped between -1 and 1, then it increments the desired joint angle $\hat{\mathbf{q}}$. Moreover, relative control is exploited, leading to (1):

$$\hat{\mathbf{q}} = clip(\hat{\mathbf{q}} + v_{max}\Delta t\mathbf{a}, \mathbf{q}_{min}, \mathbf{q}_{max}) \tag{1}$$

where Δt is the timestep and v_{max} is a constant scalar that caps the maximum speed of the joints from the policy. v_{max} is set to $7rad/s$. After the action updates the desired joint angle, it is clipped between the minimum $\mathbf{q}_{min}$ and maximum $\mathbf{q}_{max}$ joint angles of the robot hand. This control law encourages policy exploration since the maximum speed is bigger than the one physically reachable by the robot and the hand, and mimics joint torque control by using the relative speed.

3.2.4 Domain randomization

To compensate for the inaccuracy of the physics engine, make the policy more robust, and close the sim-to-real gap, domain randomization is applied to some of the system's physical properties, namely the observations, and scale, mass and friction of the cube (see *Table 3*). Domain randomization is not applied to the robotic arm's physical parameters (i.e. mass, friction, stiffness, and damping) because the robot is moved very slowly for safety reasons, and then the dynamic effects are neglected.

3.3 Training results

The results of the training are collected using Weights and Biases. Thanks to this tool, it is possible to visualize the performances of the agent. In particular, three quantities are used to evaluate the

Quantity	***Distribution***
Observation noise	$\mathcal{N}(0;\ 0.005)m$
Observations correlated noise	$\mathcal{N}(0;\ 0.005)m$
Cube friction	$\mathcal{U}(0.7;\ 1.3)$
Cube scaling	$\mathcal{U}(0.98;\ 1.02)$
Cube mass	$\mathcal{U}(0.5;\ 1.5)$

Table 3: Quantities modified by using domain randomization and correspondent noise distribution.

success of the policy:

1. the distance between the cubes' centers in the x-y plane. The success is achieved if it is lower than 0.02 meters.
2. the distance of the cubes along the z-axis. The success is achieved if it is lower than 0.02 meters (see Figure 10).
3. the distance of the hand from the center of the grabbed cube. The success is achieved if it is greater than 0.08 meters.

It is possible to visualize four quantities, useful to evaluate several policies' performances:

1. success reward: evolution of the success reward during the training.
2. success rate: percentage of instances achieving success at the end of each episode.
3. total reward: total reward achieved during the training.
4. episode length: time until which the reset condition is triggered in each episode. The reset condition is achieved either if it succeeds, if the cube is dropped from the scene, or if the maximum episode length is achieved.

The results are visualized by taking the mean of the five runs with different random seeds. Three different observation noise variance realizations are considered, as shown in the legend. An area of $\pm\sigma$ is shown around the curves.

Figure 12: Success reward. It is increasing until the agent finds a policy to succeed in the task. Then, it converges.

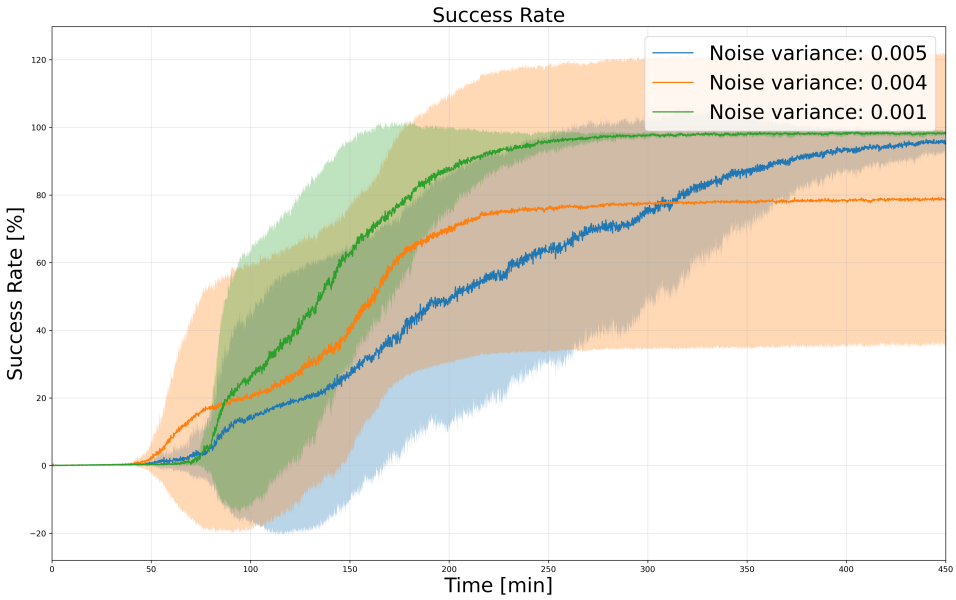


Figure 13: Success rate. When the noise variance is 0.004, one out of 5 experiments does not succeed, reducing the overall success rate. The negative success rate is due to the high standard deviation, originating from the different learning times of the different seeds.

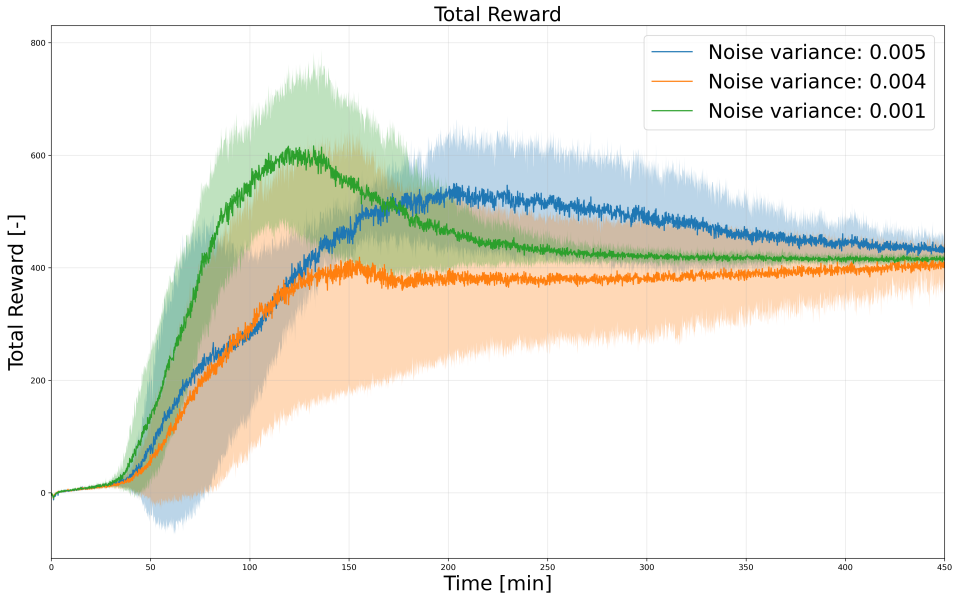


Figure 14: Total reward. It is increasing until the agent finds a policy to succeed in the task. Then, it converges.

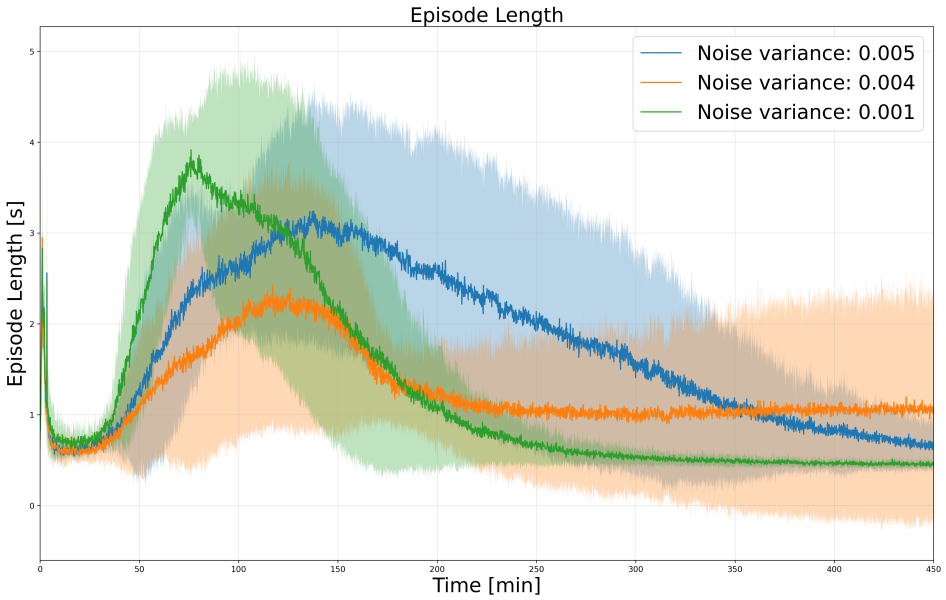


Figure 15: Episode length. It has three distinct phases. At the beginning, the average episode length is decreasing, because the agent is failing in grabbing the cube, and then, the training is constantly resetting since it triggers the reset condition. As time increases, the agent learns what is the proper way to grab, hold, and move the cube. Then, the average episode length increases. Finally, when the agent finds what is the proper policy to accomplish the task, the episode length starts decreasing as the optimization goes on.

3.4 Experiment setup

After training is finished, the neural network of the actor is exported to an ONNX format and run on the robot. For the implementation of the policy on a real-world set-up, ROS 1 Noetic has been used. A brief description of the hardware and software components is presented with more details in the following subsections.

3.4.1 Hardware architecture

In terms of hardware, there is the main robot set-up: the Faive Hand Proto2 and a Franka Emika Panda arm. There are three computers, all connected via Ethernet:

1. Franka Emika Control computer: handles low-level control of the robotic arm.
2. Intel NUC: runs ROS master, handles higher-level control of the robotic arm.
3. Desktop workstation: runs top-level tasks (cube poses tracking, logging, high-level control).

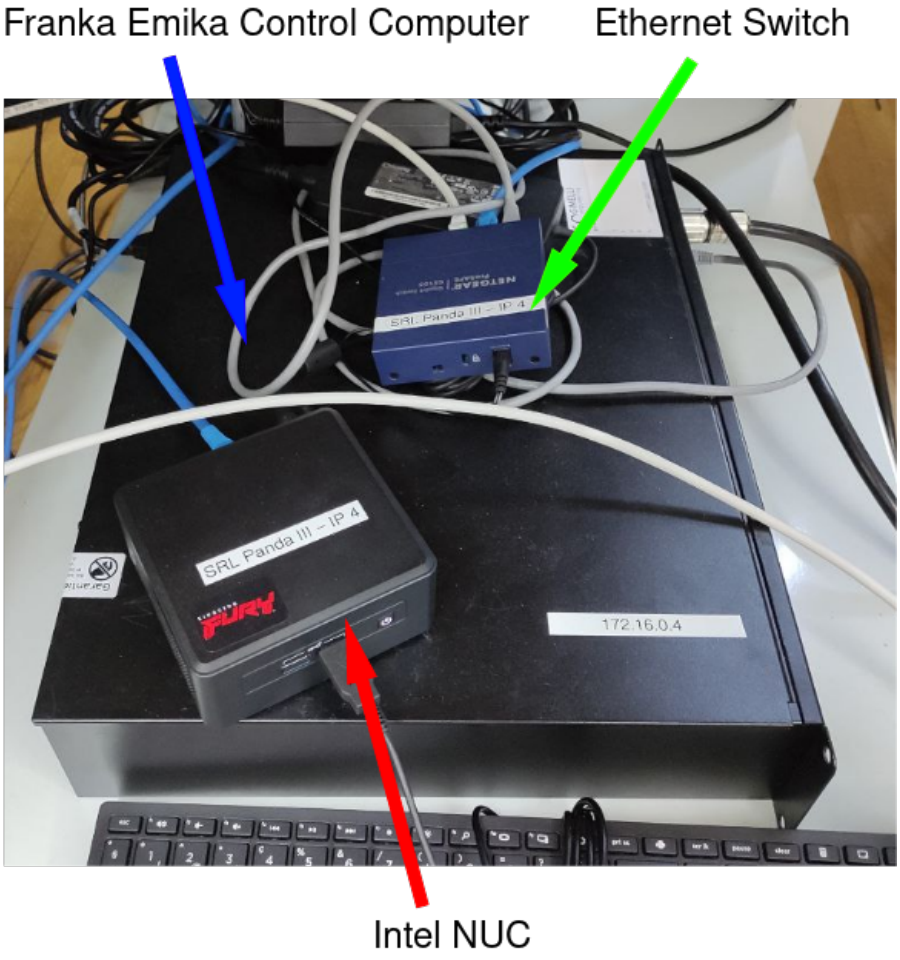


Figure 16: Hardware architecture.

Furthermore, two OAK-D Pro cameras[9] have been used (see Figure 17). The cube poses are estimated using AprilTag markers[10]. The first cube has one marker for each side, while the second one, which is the one supposed to be grabbed, has four markers for each side (see Figure 18). The reason behind this choice is to avoid possible markers' occlusions that may happen when the hand is grabbing the cube. In this way, there are more markers available to obtain the cube pose. The

[9] https://shop.luxonis.com/products/oak-d-pro?variant=42455252369631
[10] https://github.com/AprilRobotics/apriltag

estimated positions and orientations are expressed with respect to the center of the table, which is identified by placing three markers on its surface to ensure at least one of them is always visible.

Figure 17: OAK-D Pro Camera.

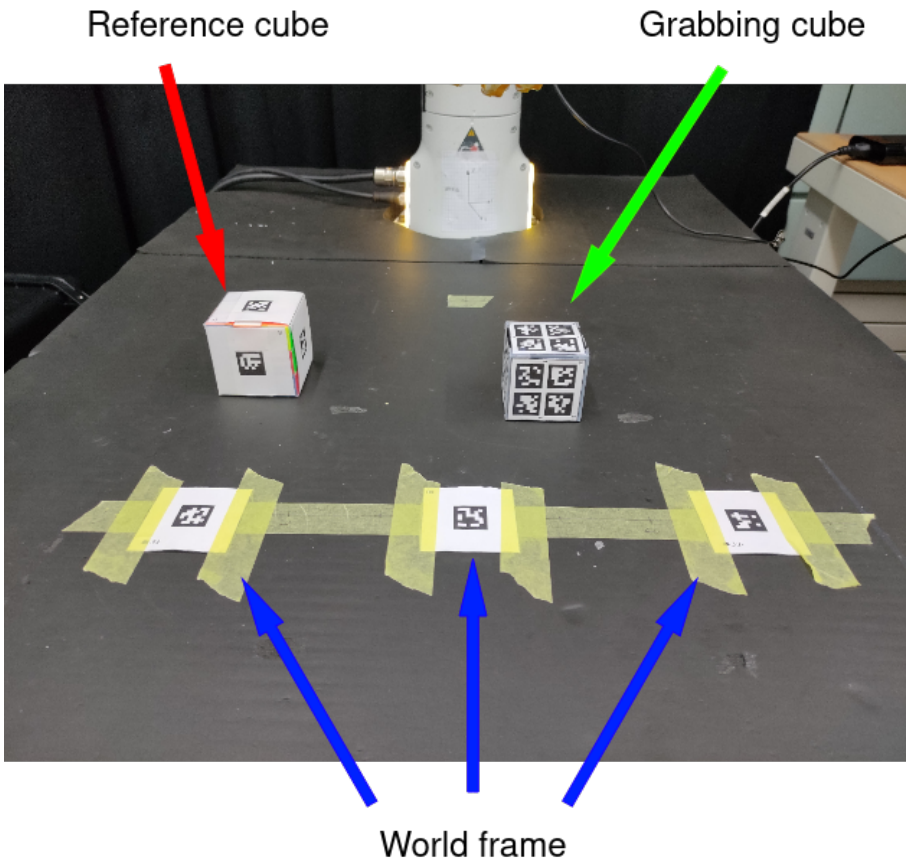


Figure 18: Objects covered with AprilTag Markers.

3.4.2 Software architecture

The software system is built around ROS Noetic and written in Python. It is structured into two main distinct blocks:

1. **pose tracking**: houses the ROS topic which publishes the messages representing the estimates of the cube poses. By using Apriltag markers, the algorithm detects the position and orientation of the objects in the scene and expresses their position directly with respect to the world frame. The markers are identified by a number that relates them to a specific object. Then, more markers identify one cube. So, it is necessary to average the poses coming from different markers to obtain the final cube poses. The data coming from the two cameras are averaged. In this way, if there are occlusions in only one of the cameras, the algorithm is still able to estimate the pose from the data coming from the other camera.
2. **control policy**: houses the ROS topic which publishes the messages containing the control commands to the robotic arm and the Faive Hand. The control policy is inferred based on the observations read from the ROS messages, i.e. joint arm angles, joint hand angles, end-effector pose, and cubes poses. The relative control speed of the robot is slowed down with respect to the one used during training to avoid crashing into the table and breaking the hand.

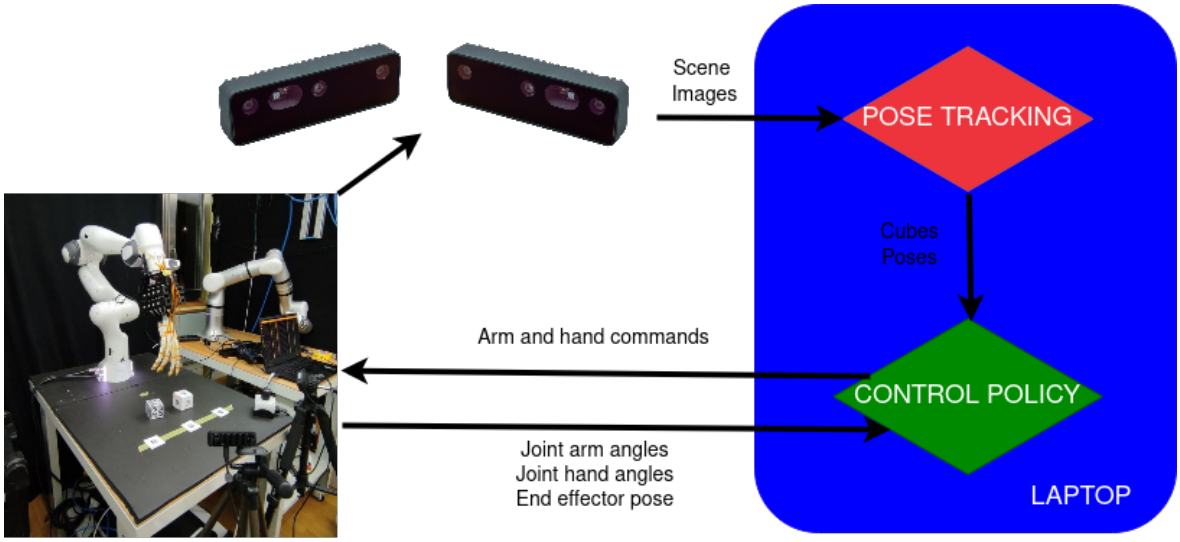


Figure 19: Control flow diagram.

3.5 Experiments results

Several experiments have been run to evaluate the performances of the agent in a real-world environment. All the policies that were achieving success have been tested on the real robot. Several issues were encountered, in particular, related to the motion constraints that are acting on the robot to guarantee a safe movement of the robot with respect to itself and its surroundings (see Figure 20). More in detail, three major constraints are limiting the robot's movement:

1. self-collision avoidance, i.e. avoiding collision within the links of the robot.
2. bounding-box, i.e. limiting the robot's workspace by manually defining limits within it. In particular, the constraints are expressed with respect to the end-effector position.
3. environment-collision avoidance, i.e. avoiding collision with the table.

These elements were not considered during the training of the agent in Isaac Gym, and then, deeply affected the final policy (see Figure 21).

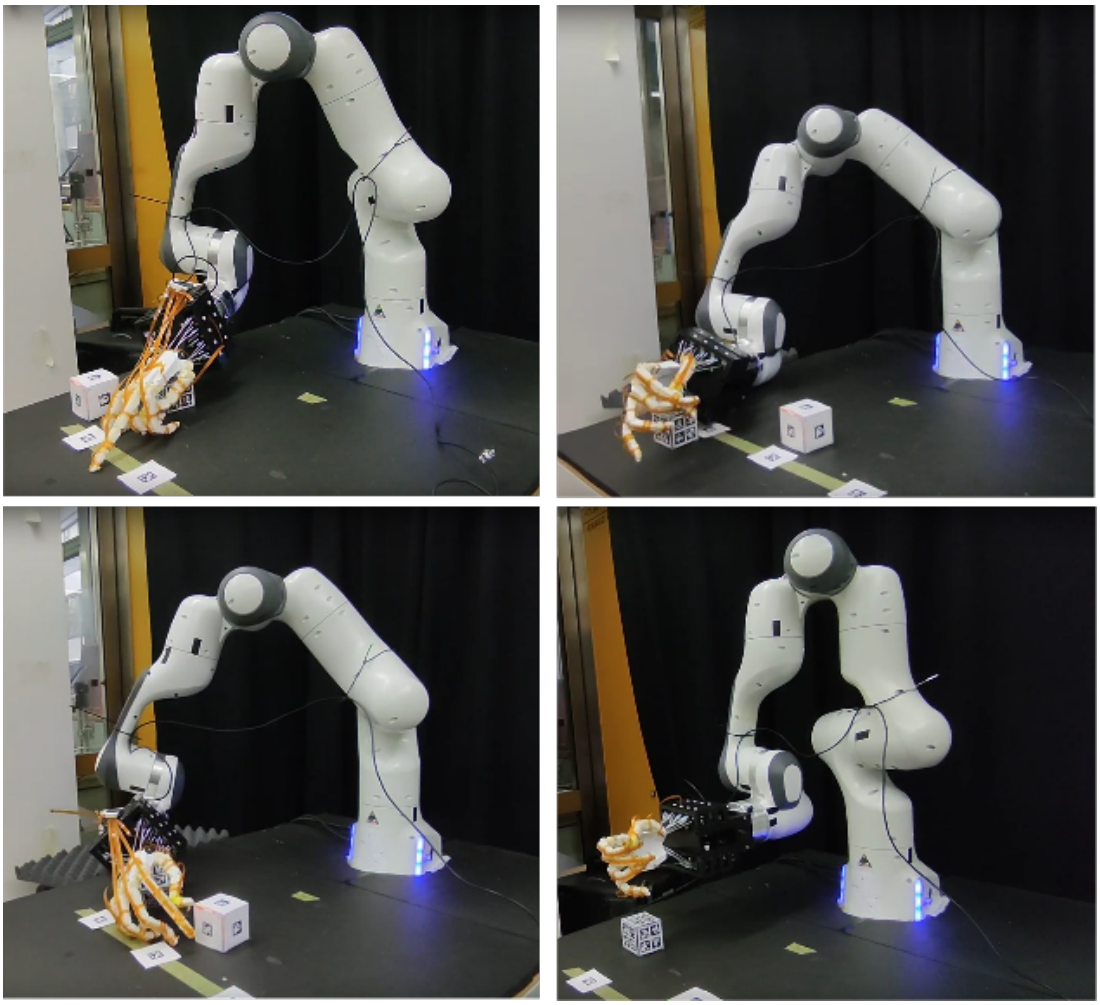


Figure 20: Pictures of the robot blocked due to collisions while running different policies.

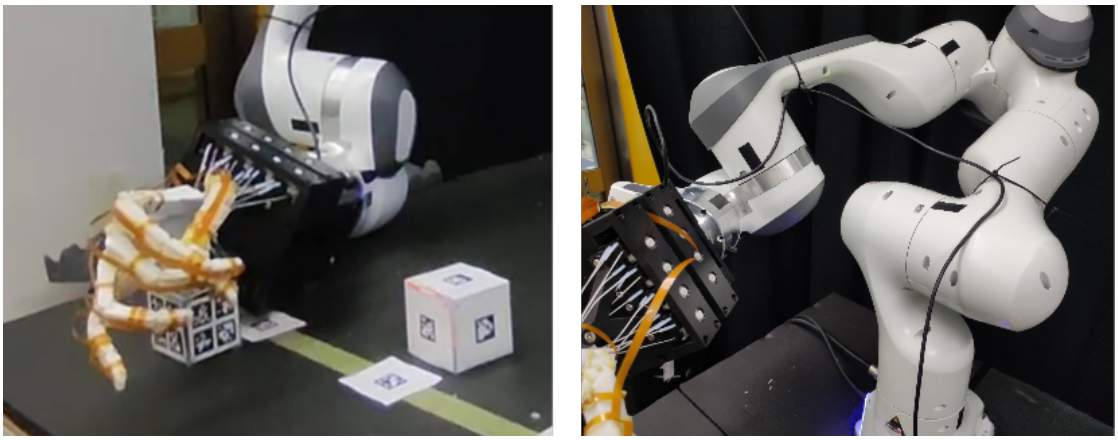


Figure 21: Detailed pictures of the robot blocked due to collisions. In the left image, the robotic arm is touching the table, while in the right image, two links of the robotic arm are too close to each other.

As it is possible to see in the previous sections, the trainings are achieving a high success rate and promising performances. Nevertheless, the experiments performed in the laboratory showed that improvements are needed to transfer the learned policy to the real world. On one hand, considering the whole experimental setup, several blocks are separately working (see Figure 22):

1. the hand approaches the correct cube, i.e. the one which the agent is trained to grab.
2. the hand holds the cube when it is manually placed in its proximity.
3. following from the two previous achievements, the cube tracking algorithm is properly detecting the cubes' poses.

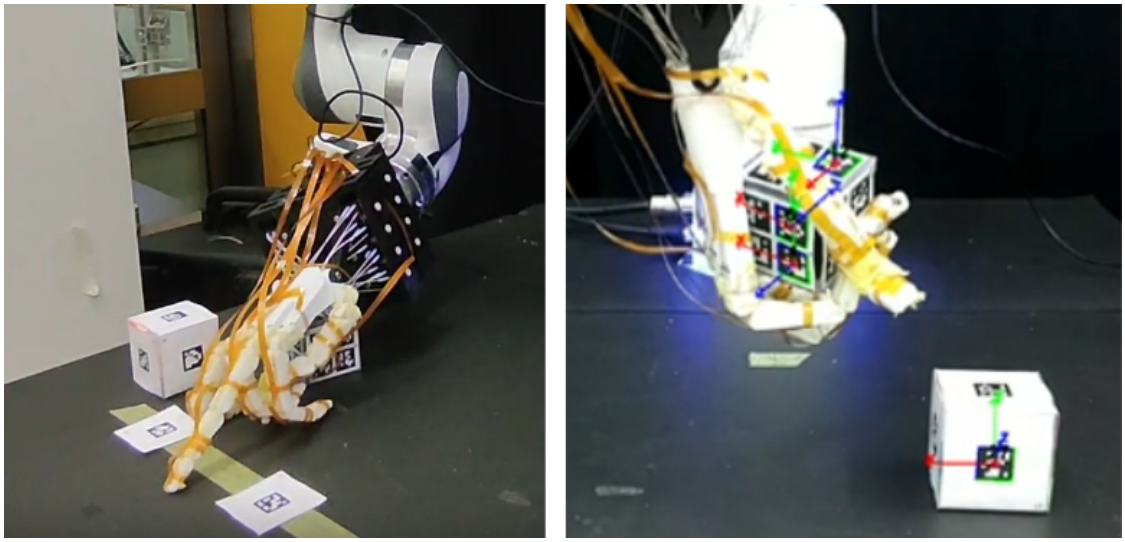


Figure 22: Detailed pictures of the robot. In the left image, the robotic arm is getting closer to the grabbing cube, while in the right image, the hand is holding the grabbing cube.

On the other hand, several components are not working correctly:

1. the arm is colliding with itself and with the environment.
2. the hand is not grabbing the cube after reaching it.
3. the cubes are not stacked at the end, which was the desired goal.

4 Conclusion

This section describes the overall results of this Semester's Thesis, and identifies future improvements that are worth implementing.

4.1 Overall results

The training performed in Isaac Gym achieves consistent and relevant results. In particular, even if different seeds and noise realizations are used, the performances are satisfying. More in detail, the success rate and the episode length are very promising, since at least 80% of the instances are completing the task in less than one second. These suggest that transferring the policies to the real world should be possible, but challenging as it is in many other applications.
Nevertheless, the experiments showed that there are still several elements to fix before closing the sim-to-real gap. Even if some of the components of the control pipeline are working separately, for example, the tracking of the position and orientation of the cubes, others are not behaving as expected. In particular, several constraints on the robot's movements were not taken into account when setting up the simulation environment, causing major problems when testing the policies in the real world.

4.2 Future work

Transferring the policy learnt in simulation to the real world is the main challenge of this Semester Thesis. When moving from simulation to reality, many issues are encountered, for example, dealing with the inherent noise that affects every measurement, or avoiding breaking the hardware. Several improvements are possible for the current setup:

1. considering the current design of the Faive Hand Proto2, since no measurements coming directly from the hand are used as observations for the reinforcement learning training, no additional sensors are needed. At the same time, covering the fingertips with silicone may make it easier to grasp the cubes, or other objects used in other applications.
2. introducing explicitly the constraints on the robots, for example, the self collisions or the collision with the table, as penalties in the simulation, otherwise the robot will always stop when running the policy. This issue may be solved by designing custom penalties for the robot's joint positions with respect to their surroundings.
3. the train in Isaac Gym lasts 4000 epochs. Completing one full training took, according to Weights & Biases, more than 8 hours. Considering that the training in [8] used the same number of epochs, but the training lasted only a couple of hours, some improvements on the simulation are needed. In particular, the biggest computational effort should be in generating the video frames on Weights & Biases used to visually evaluate the policy.
4. improves the cubes detection algorithm, since the markers may get occluded and the measurements are affected by high noise. Different marker sizes can be exploited, and different placements of the cameras should be considered.
5. introduces domain randomization on more elements, for example, the dynamical parameters of the robotics arm, or the actions. These additional elements might alleviate the difficulties in transferring the policy into the real world.

References

[1] Yashraj Narang et al. *Factory: Fast Contact for Robotic Assembly*. 2022. arXiv: 2205.03532 [cs.RO].

[2] A.M. Okamura, N. Smaby, and M.R. Cutkosky. "An overview of dexterous manipulation." In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 1. 2000, 255–262 vol.1. DOI: 10.1109/ROBOT.2000.844067.

[3] Chunmiao Yu and Peng Wang. "Dexterous Manipulation for Multi-Fingered Robotic Hands With Reinforcement Learning: A Review." In: *Frontiers in Neurorobotics* 16 (2022). ISSN: 1662-5218. DOI: 10.3389/fnbot.2022.861825. URL: https://www.frontiersin.org/articles/10.3389/fnbot.2022.861825.

[4] Stephen C. Jacobsen et al. "Design of the Utah/M.I.T. Dextrous Hand." In: *Proceedings of the 1986 IEEE International Conference on Robotics and Automation, San Francisco, California, USA, April 7-10, 1986*. IEEE, 1986, pp. 1520–1532. ISBN: 0-8186-0695-9. DOI: 10.1109/ROBOT.1986.1087395. URL: http://dx.doi.org/10.1109/ROBOT.1986.1087395.

[5] Hong Liu et al. "Multisensory five-finger dexterous hand: The DLR/HIT Hand II." In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2008), pp. 3692–3697. URL: https://api.semanticscholar.org/CorpusID:5965566.

[6] OpenAI Andrychowicz et al. "Learning dexterous in-hand manipulation." In: *The International Journal of Robotics Research* 39 (Nov. 2019), p. 027836491988744. DOI: 10.1177/0278364919887447.

[7] Michel Breyer et al. *Comparing Task Simplifications to Learn Closed-Loop Object Picking Using Deep Reinforcement Learning*. 2019. arXiv: 1803.04996 [cs.RO].

[8] Yasunori Toshimitsu et al. *Getting the Ball Rolling: Learning a Dexterous Policy for a Biomimetic Tendon-Driven Hand with Rolling Contact Joints*. 2023. arXiv: 2308.02453 [cs.RO].

[9] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[10] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control." In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[11] Viktor Makoviychuk et al. *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*. 2021. arXiv: 2108.10470 [cs.RO].