2023 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)
June 28-30, 2023. Seattle, Washington, USA

A Supervisory Learning Control Framework for Autonomous & Real-time Task Planning for an Underactuated Cooperative Robotic task

Sander De Witte[1,2], Tom Lefebvre[1,2], Thijs Van Hauwermeiren[1,2] and Guillaume Crevecoeur[1,2]

Abstract— **We introduce a framework for cooperative manipulation, applied on an underactuated manipulation problem. Two stationary robotic manipulators are required to cooperate in order to reposition an object within their shared work space. Control of multi-agent systems for manipulation tasks cannot rely on individual control strategies with little to no communication between the agents that serve the common objective through swarming. Instead a coordination strategy is required that queries subtasks to the individual agents. We formulate the problem in a Task And Motion Planning (TAMP) setting, while considering a decomposition strategy that allows us to treat the task and motion planning problems separately. We solve the supervisory planning problem offline using deep Reinforcement Learning techniques resulting into a supervisory policy capable of coordinating the two manipulators into a successful execution of the pick-and-place task. Additionally, a benefit of solving the task planning problem offline is the possibility of real-time (re)planning, demonstrating robustness in the event of subtask execution failure or on-the-fly task changes. The framework achieved zero-shot deployment on the real setup with a success rate that is higher than 90%.**

I. INTRODUCTION

Cooperative robots are an increasingly important aspect of modern robotics, with applications ranging from unmanned vehicles to robotic manipulators [1]–[3]. Cooperative robots are able to work together to serve a common goal, leveraging their combined capabilities to perform tasks that would be impossible for a single robot to accomplish. For example, individual agents are limited in performing an assembly task on their own; these limitations can be overcome by using a cooperative set-up of robotic manipulators [4]. Cooperation results in the ability to perform tasks with a higher complexity, by increasing the manipulation space of a single robot by using multiple robots. In this study, we examine how two robotic manipulators can collaborate to complete complex tasks by sharing a portion of their individual workspace. The manipulators are tasked with moving an object to a desired location and orientation within the global workspace. By working together, the manipulators can achieve these goals through multiple, collaborative actions in the shared workspace, illustrated in Fig. 1.

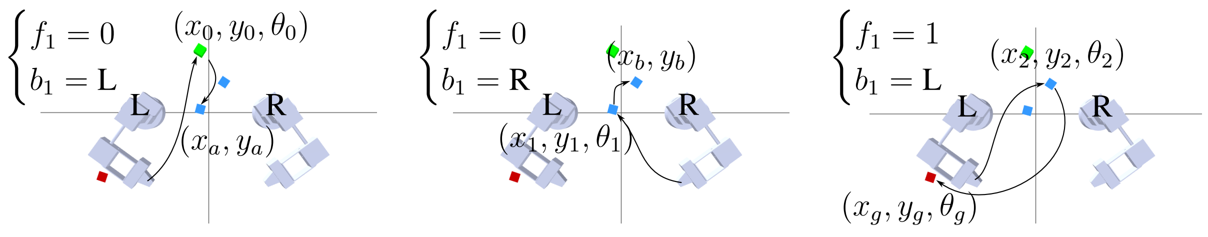
Task and Motion Planning encompass the solving of combined problems involving a finite sequence of discrete mode types, e.g. which objects to pick and place, continuous mode parameters, the poses and grasps associated with the movable objects, and the continuous motion paths [5]. Standard TAMP approaches perform a combined optimization of all these parameters, handling the interdependence of the motion-level and the task-level aspects of the problem. Cooperation between different robots performing a single task, can be described as a TAMP problem. Which robot to query to perform a certain action is then an example of an extra discrete mode type. In addition we can look at the above mentioned cooperation problem as such a problem. In this approach however, the problem is decomposed in separate scheduling and motion planning problems. The discrete decisions are completely disconnected from the continuous motion. The next best task, a new pick-and-place action with its associated grasp poses for one of the robots, is determined without performing a trajectory optimization for each action, resulting in a simplified optimization problem.

To cope with changes in the environment, such as changes in the desired goal configuration or difficulties with individual subtasks, we aim to develop a parametrized policy, to determine the next best action, that only depends on the current object configuration and the target goal configuration. The policy will determine the optimal sequence of actions, including which manipulator should perform each manipulation, to reach the desired goal. Real-time computation benefits from this control decomposition in supervisory control and low-level control, as the associated optimization problem is significantly reduced.

The found policy could be used to accelerate interleaved TAMP approaches as well, as is shown in other research, that apply learning techniques to accelerate TAMP. Driess et al. learn a classifier to determine the feasibility of the resulting geometric problem for a discrete decision based on an image of the workspace [6]. In the research by Chitnis et al. [7] learning methods are used to guide a combined search to find high-level symbolic plans and their low-level refinements, e.g. the associated motion plans and grasp poses. At the low-level they learn how to refine the high-level plans, by learning to propose continuous values for symbolic references that are likely to result in collision-free trajectories. In a similar vein, our method could be used to output the next best action that has a feasible trajectory, thus guiding the search for an optimal action sequence.

Our approach uses a reinforcement learning approach to train this parametrized policy. Reinforcement learning is a machine learning technique that allows agents to learn from their experiences and improve their decision-making over time. Deep reinforcement learning has led to a number of successes in learning policies for sequential decision-making problems in both simulated environments [8], [9] as robotic tasks [10], [11]. Translation of policies trained in simulation to the real world poses a major problem of current RL

[1]Department of Electromechanical, Systems and Metal Engineering, Ghent University, Tech Lane Ghent Science Park 131, B-9052 Zwijnaarde, Belgium
[2]Core Lab MIRO, Flanders Make Strategic Research Centre for the Manufacturing Industry



$\begin{cases} f_1 = 0 \\ b_1 = \text{L} \end{cases}$
(x_0, y_0, θ_0)
L
R
(x_a, y_a)
$\begin{cases} f_1 = 0 \\ b_1 = \text{R} \end{cases}$
(x_b, y_b)
L
R
(x_1, y_1, θ_1)
$\begin{cases} f_1 = 1 \\ b_1 = \text{L} \end{cases}$
(x_2, y_2, θ_2)
L
R
(x_g, y_g, θ_g)

(a) First action $a_1 = (0, L, x_a, y_a)$ (b) Second action $a_2 = (0, R, x_b, y_b)$ (c) Final action $a_3 = (1, L, x_g, y_g)$

Fig. 1: Schematic figure showing multiple collaborative actions to pick-and-place object to its desired orientation, which is unachievable by an individual manipulator because it has only five DoFs.

research in robotics. The latter forms less of a problem in our approach, since the learning is performed on a higher level.

Using a discretized version of the shared workspace, from which the next best action is chosen works [12], but results in a low success rate and is difficult to train. This could be circumvented by using a continuous approximation of the shared workspace, resulting in a more complete parametrization of the shared workspace. The parametrization is no longer limited to a set of discrete points in the shared workspace, but comprises it completely. The motivating question is if the continuous parametrization results in better results, such as an increased training speed and a better success rate.

II. Problem statement

We start by describing the small-scale setup and its associated manipulation problem. The setup consist out of a work cell equipped with two low-cost plane mirrored manipulators and a perception system. The perception system consist of a camera attached at the top of the work cell with a clear top-down view of the workspace and is able to identify and locate objects. Both manipulators have five degrees of freedom (DoFs) and are equipped with a low-level controller, capable of pick-and-placing an object in the base xy-plane. This plane will be referred to as the manipulation plane.

Two DoFs are used to control the pitch and roll of the end-effector, during pick-up and drop-off these need to be zero to ensure a correct grasp and release. Two DoFs are used to control the position in the manipulation plane, while the last DoF is used to control the height. Therefore, we can not directly control the yaw of the robot's end-effector and consequently the yaw of the object to manipulate, henceforth simply referred to as the object's orientation. This results in an underactuated manipulation problem.

The final orientation will be the result of consecutive pick-and-place maneuvers performed by each robot, more specifically the pick-up and drop-off positions and the geometry of the manipulators. The individual manipulation planes of the left and right (respectively L and R) robots are referred to as $\mathcal{M}_\text{L} \subset \mathbb{R}^2$ and $\mathcal{M}_\text{R} \subset \mathbb{R}^2$, which are defined as the controllable set of end-effector configurations in the manipulation plane expressed as Cartesian coordinates with respect to a global frame of reference. In other words, the individual manipulation spaces are determined as the intersection of the individual manipulator workspaces, expressed in the global frame of reference, and the manipulation plane. These feasible workspaces consist out of the collection of points that are collision-free, including self-collision and collision with the ground. Additionally we can find the global $\mathcal{M}$ and the mutual $\mathcal{M}_m$ manipulation spaces, respectively, as the union and intersection of the individual workspaces.

$$\mathcal{M} = \mathcal{M}_\text{L} \cup \mathcal{M}_\text{R}$$
$$\mathcal{M}_m = \mathcal{M}_\text{L} \cap \mathcal{M}_\text{R}$$

A pick-and-place task is considered where objects, placed in the global manipulation space, need to be repositioned to a new and given goal position $(x_g, y_g) \in \mathcal{M}$ and desired orientation θ_g. By adding this orientation to the manipulation space, the state space $\mathcal{S} = \mathcal{M} \times \mathbb{R}$ is obtained. A successful manipulation is achieved if the object has the desired position and orientation. When for example the goal position $\mathbf{x}_g \in \mathcal{M}_\text{L}$ and the present position $\mathbf{x}_t \in \mathcal{M}_\text{R}$, it is clear that the task can only be achieved by a hand over in the mutual work space. Furthermore considering that the control of the individual manipulators is such that they can not control the orientation of the object, a sequence of hand-overs will be required to maneuver the desired object into its desired configuration $\mathbf{s}_g = (x_g, y_g, \theta_g) \in \mathcal{S} = \mathcal{M} \times \mathbb{R}$. In particular a sequence of varying hand-over positions must be determined in a way that the subsequent relative reorientations accumulate into the desired final orientation.

The objective of this work is therefore to find a control strategy capable of querying a sequence of isolated pick-and-place maneuvers, each of which can be executed by a single manipulator, and, so that such a sequence exists from any arbitrary starting configuration to any desired goal configuration that intersects with the global manipulation space.

III. Methodology

The control problem described in section II qualifies as a TAMP problem since both the optimal hand-over sequence can be determined as well as the individual motions in between each hand-over. In this work we propose a hierarchical control strategy decomposing the TAMP problem into separate scheduling and motion planning problems.

A. Mathematical problem definition

The state of the system, indicated by $\mathbf{s}_t$, consist out of the Cartesian coordinates of the manipulated object and its orientation. Similarly, the goal configuration of the object is denoted as $\mathbf{s}_g$.

$$\begin{aligned}\mathbf{s}_t &= (x_t, y_t, \theta_t) \in \mathcal{S} \\ \mathbf{s}_g &= (x_g, y_g, \theta_g) \in \mathcal{S}\end{aligned}$$

The objective is to design a policy function $\pi : \mathcal{S}^2 \mapsto \mathcal{A}$, so that for any initial state in the state space $s_0 \in \mathcal{S}$, a finite sequence of pick-and-place actions $\mathbf{a}_t \in \mathcal{A}$ can be determined such that the object can be manipulated into any desired goal configuration in the observation space $\mathbf{s}_g \in \mathcal{S}$. We will detail the definition of the action in the next paragraph. For the moment it is assumed that the action contains sufficient information for the system to change the state of the object from $\mathbf{s}_t$ to $\mathbf{s}_{t+1}$. This problem can be then formulated as a first-exit optimal control problem.

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\mathbf{s}_0 \sim \mathcal{U}(\mathcal{S})} \left[\sum\nolimits_{t=0}^{t'} r(\mathbf{s}_t, \mathbf{s}_g, \pi(\mathbf{s}_t, \mathbf{s}_g)) \right] \quad (1)$$

Here $\mathcal{U}(\mathcal{S})$ denotes the uniform distribution on $\mathcal{S}$. The reward $r : \mathcal{S}^2 \times \mathcal{A} \mapsto \mathbb{R}$ can be used to express different abstract features of the policy such as maximum accuracy or minimal execution time or can directly relate to the underlying motion optimization problem. We discuss this possibility in the final paragraph. Finally, since a first exit-optimal control problem is considered with a time independent reward function, the optimal policy function will be time-invariant.

Let us now characterize the information that is encoded in to the action, $\mathbf{a}_t$. The action consists out of three separate decisions:

1) whether the present action is final, indicated by the Boolean $f_t \in 0, 1$;
2) whether the left or right manipulator needs to be queried, given by the discrete variable $b_t \in \{\text{L}, \text{R}\}$;
3) the next drop-off position, represented by $(x_{t+1}, y_{t+1}) \in \mathcal{M}$ in Cartesian coordinates.

Formally, the following action space is retrieved:

$$\mathbf{a}_t = (f_t, b_t, x_{t+1}, y_{t+1}) \in \mathcal{A} = \{0, 1\} \times \{\text{L}, \text{R}\} \times \mathcal{M} \quad (2)$$

Presented with this information the selected manipulator should be able to determine and execute the motion $(x_t, y_t) \to (x_{t+1}, y_{t+1})$. Some further rationalizations can be imposed on the policy. In practice the drop-off position is limited to the mutual manipulation space $(x_t, y_t) \in \mathcal{M}_m$ when $f_t = 0$, as this is the only space both agents can reach. Only in the final step, $f_t = 1$, the action can coincide with the global manipulation space, as the drop-off position should coincide with the goal position. Which agent to query is not straightforward, since the goal state's location can be anywhere in the global manipulation space. Though it may seem rational to alternate between the two manipulators in between every query, taking into account the possibility that the manipulator does not successfully execute the motion $(x_t, y_t) \to (x_{t+1}, y_{t+1})$. We opt to leave this decision to the policy rather than imposing alternation. Additionally the goal configuration could also change mid-execution.

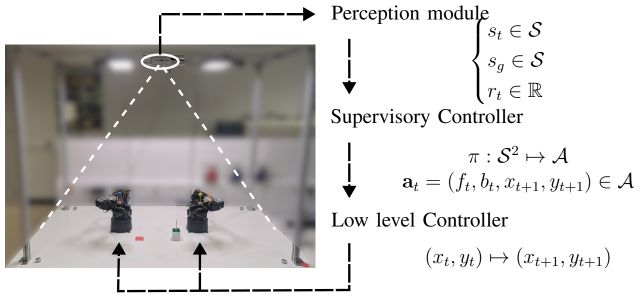

Perception module
$s_t \in \mathcal{S}$
$s_g \in \mathcal{S}$
$r_t \in \mathbb{R}$
Supervisory Controller
$\pi : \mathcal{S}^2 \mapsto \mathcal{A}$
$\mathbf{a}_t = (f_t, b_t, x_{t+1}, y_{t+1}) \in \mathcal{A}$
Low level Controller
$(x_t, y_t) \mapsto (x_{t+1}, y_{t+1})$

Fig. 2: Overview of the method, showing clear decomposition between supervisory controller and low level controller.

The state update rule is determined by a nonlinear function, that depends on the forward kinematics of the queried robotic manipulator, and does not depend on the motion plan between two positions, since the relative orientation between the object and the robot is assumed to be constant.

$$\begin{aligned}\mathbf{s}_{t+1} &= \mathbf{f}(\mathbf{s}_t, \mathbf{s}_g, \mathbf{a}_t) \\ &= (x_{t+1}(\mathbf{s}_g, \mathbf{a}_t), y_{t+1}(\mathbf{s}_g, \mathbf{a}_t), \theta(\mathbf{s}_t, \mathbf{s}_g, \mathbf{a}_t))\end{aligned}$$

In conclusion we may note that the low level motion plan, i.e. continuous manipulator path between the two positions and $(x_t, y_t) \to (x_{t+1}, y_{t+1})$, is determined and performed by the individual manipulators themselves. It is also possible that the motion plan is optimal and maximizes some reward. In the most general case this means that the scheduling reward can be represented as an underlying path planning problem.

$$\begin{aligned} r(\mathbf{s}_t, \mathbf{s}_g, \mathbf{a}_t) &= \min_{\mathbf{q}(\tau)} \int_0^1 c(\mathbf{q}(\tau), \mathbf{s}_g) d\tau \\ \text{s.t. } \mathbf{p}_t &= \mathcal{F}_{b_t}(\mathbf{q}(0)) \\ \mathbf{p}_{t+1} &= \mathcal{F}_{b_t}(\mathbf{q}(1)) \end{aligned} \quad (3)$$

A full TAMP approach would thus require to solve problem (1) and (3) simultaneously. In this work we decouple both problems completely, neglecting the possible influence of the motion planning problem on the task scheduling. However for this influence to have any significant effect on the solution, the motion planning objective should be represented in the scheduling problem. In this work we focus on sparse rewards, hence validating the assumption.

B. Solution strategy

In this section we discuss how we solve problem (1). Our solution approach requires us to parametrize the action and thus in particular the mutual manipulation space, $\mathcal{M}$. Second we discuss a parametrization strategy for the optimal policy, π^*. Finally we discuss a strategy to find the optimal policy by learning optimal parameters.

*1) **Parametrization of the solution space:*** As mentioned previously, we can limit the drop-off position to the mutual manipulation space. The feasible positions in the manipulation plane for each manipulator lie in between two concentric circles, with a minimum radius and maximum radius, $\underline{r}$ and $\overline{r}$ respectively. Since the set-up is plane symmetric

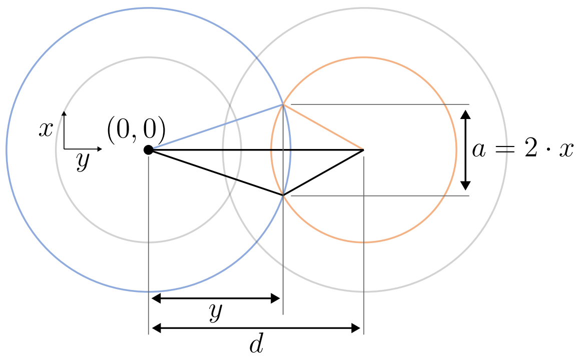


Fig. 3: Visual representation of the continuous representation.

(or mirrored), these are the same for each robot. This is visualized in Fig. 3.

We can parametrize the mutual workspace with two variables, $\alpha \in [-1,1]$ and $\beta \in [-1,1]$, representing the x- and y-position respectively. We can normalize the x-position between its maximum, $\overline{x}$, and minimum, $\underline{x}$, which is the intersection of the two large circles. Since $\underline{x} = -\overline{x}$ we can scale α with this maximum value to retrieve x.

$$x = \alpha \cdot \overline{x}$$

with

$$\overline{x} = \sqrt{\overline{r}^2 - d^2/4}$$

Moving up from $\underline{x}$ to $\overline{x}$, the width of the mutual workspace changes. Therefore the maximum and minimum y-value, respectively $\overline{y}$ and $\underline{y}$, vary with x. The width either correspond to the smaller, inside the bounds $[x', -x']$, or larger circle, outside the bounds. The value where the width changes can be found as the intersection between the blue and red circle, as shown in Fig. 3.

$$x' = \sqrt{\frac{4d^2R^2 - (d^2 - r^2 + R^2)^2}{4d^2}}$$

Given a certain x, y can be retrieved from β as:

$$y = \beta \cdot \overline{y}$$

with

$$\overline{y} = \begin{cases} \sqrt{\underline{r}^2 - |x|^2}, & x \in [-x', x'] \\ \sqrt{\overline{r}^2 - |x|^2}, & x \in [-\overline{x}, -x'] \cup [x', \overline{x}] \end{cases}$$

and assuming again, due to the symmetry, that $\underline{y} = -\overline{y}$.

2) ***Policy definition***: We opt to find a parametrized policy $\pi(\cdot,\cdot) \approx \pi(\cdot,\cdot;\phi)$, which assigns a subtask at each discrete time step to one of the manipulators. These perform their own low-level motion plan based on this given subtask. This means that instead of solving problem (1) for a specific initial state each time, we aim to solve for a global policy representation. This has the advantage that once the policy has been learning, the policy can be queried explicitly saving essential computational resources in a real-time setting.

Since the policy is now an explicit function representation, mapping the state, $\mathbf{s}_t$, to the corresponding optimal action, $\mathbf{a}_t$, it is possible to expand the input space with additional features that may encode useful information to build the highly nonlinear mapping. Therefore, as we want to implement the dependency on the changing goal, we include the goal information in the state, resulting in an augmented state and a contextual policy. The previous robot that performed the action is included as well, this to nudge the policy in the direction of switching between robots.

$$\hat{\mathbf{s}}_t = (x_t, y_t, \theta_t, x_g, y_g, \theta_g, b_{t-1}) \in \mathcal{S}^2 \times \{\mathrm{L}, \mathrm{R}\} \qquad (4)$$

Two different policy architectures are considered. One where the policy defines the final step and one where this final step is determined externally, using the kinematics of both manipulators.

a) Architecture A: One of the main complexities in the problem arises from performing the final step. The final action is defined by the Boolean f_t and affects the problem in a highly nonlinear fashion. When eliminating this Boolean from the action, after each step it needs to be checked if the object in the mutual manipulation space corresponds with the desired goal configuration. As the complete goal information is provided in the augmented state, this check can be performed for both manipulators. This check is performed by placing the block at the goal position in simulation and check if the desired orientation is achieved. If this orientation is achieved for one of the two manipulators the object performs the final step.

Because the final step is no longer dictated by the policy the action space reduces to

$$\hat{\mathbf{a}}_t = (b_t, \alpha_t, \beta_t) \in \mathcal{A}' = \{\mathrm{L}, \mathrm{R}\} \times [0,1]^2$$

where α_t and β_t represent x_{t+1} and y_{t+1} respectively.

The policy now has to find a sequence of actions resulting in a state in the mutual manipulation space that correspond to the goal configuration. The policy, however, has no idea what robot performs the final step as this is dictated externally.

b) Architecture B: If the policy dictates the final step, it has to determine which manipulator needs to perform it and when it needs to performed. It has to dictate which state in the mutual manipulation space corresponds to which goal state for each possible manipulator, while also deciding the sequence of actions to get to that state. The final step is now a part of the action sequence. The state is unaltered from equation 4, while action 2 is adapted to include the continuous parametrization.

$$\hat{\mathbf{a}}_t = (f_t, b_t, \alpha_t, \beta_t) \in \mathcal{A}' = \{0,1\} \times \{\mathrm{L}, \mathrm{R}\} \times [0,1]^2$$

3) ***Reinforcement learning***: We learn this policy using reinforcement learning. The solution space is parametrized using continuous variables, therefore it is opted to go with an actor-critic method, that concurrently learns a Q-function and a policy.

a) Learning algorithm: We opted to go for Soft Actor Critic, as this is a popular choice and provided a stable implementation. Although other methods could have worked as well, such as TD3 [13] or PPO [14]. SAC optimizes a stochastic policy in an off-policy way, with as main feature the use of entropy regularization. It is trained to maximize a trade-off between expected return and entropy. Entropy is a measure of the randomness in the system. If x is a random value, with a probability mass P. The entropy H of x can be distributed from the its distribution P according to

$$H(P) = \mathop{\mathbb{E}}_{x \sim P}(-\log P(x))$$

The agent will get a bonus reward each time step proportional to the entropy of the policy at that timestep. Increasing entropy results in more exploration, which can consequently improve learning later on. Additionally, converging to a bad local optimum can be prevented. The problem description changes to

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau \sim \pi} \sum_{t=0}^{\infty} \gamma^t \big(R(s_t, a_t, s_{t+1}) + \alpha H \left(\pi(\cdot|s_t) \right) \big)$$

Further details can be found in the corresponding literature [15].

b) Reward shaping: In order to minimize manual tuning we opted for sparse rewards for both architectures, with no extra information being encoded in the reward. For the first architecture this is rather easy, since we have only two scenarios. If the state of the object in the mutual workspace correspond to a desired goal state for one of the two manipulators, the episode is terminated and the object is placed at its goal, without receiving a negative reward. In the other case, where the state is not as desired, the episode continues and a reward of minus one is returned. This way the episode length is minimized.

$$r_A = \begin{cases} 0, & |\theta_t^{b_{t+1}} - \theta_g| \leq \epsilon \\ -1, & |\theta_t^{b_{t+1}} - \theta_g| > \epsilon \end{cases}$$

with $\theta_t^{b_{t+1}}$ indicating the orientation at the goal position, placed there by robot b_{t+1}.

For architecture B however new ways for an episode to fail are introduced. In architecture B, since the policy dictates when to place the object at the desired position, the object can be placed at the final position at the wrong time. Additionally, the policy can query the wrong robot, resulting in either a wrong orientation or an infeasible goal position. Finally, the policy could never make the decision to take the final step.

After some minor manual tuning the following sparse reward function is retrieved

$$r_B = \begin{cases} 20, & (x_t, y_t) \equiv (x_g, y_g) \text{ and } |\theta - \theta_g| \leq \epsilon \\ -1, & (x_t, y_t) \in \mathcal{M}_m \\ -30, & (x_t, y_t) \equiv (x_g, y_g) \text{ and } |\theta - \theta_g| > \epsilon \\ & \text{or } (x_t, y_t) \vee (x_{t-1}, y_{t-1}) \notin \mathcal{M}_{b_t} \end{cases}$$

with $\mathcal{M}_{b_t}$ indicating the workspace of the robot performing the action at t. The final case indicates when the robot needs to pick-up the object outside its manipulation space, which is infeasible. To avoid ending up in a local minimum, where the object is placed directly at the goal, resulting in a reward of -1, a positive reward is given if the goal is achieved. Additionally, a strong negative reward is returned for infeasible actions.

c) Hindsight experience replay: Since we are dealing with sparse rewards, when the block is misplaced the reward provides no additional information except that the performed sequence of actions does not lead to a successful goal configuration. It is however possible to act as if the found configuration was the desired goal, which gives us extra information. This is exactly what Hindsight Experience Replay does, for the specific details we refer to [16].

d) Further remarks: SAC is used to train both architectures. In the first the action has three dimensions, while the action has four dimensions in architecture B. The latter is harder to learn, therefore a simplified form of curriculum learning [17] is implemented. The policy is first trained on a simpler environment where the allowed mistake on the orientation is larger. This gives the policy an incentive to move to the goal position faster, which helps training. By gradually increasing the difficulty by making the allowed error smaller, the policy learns eventually in the correct environment. In practice two training steps are enough in order to get to the desired accuracy.

In order to train the policy, a significant amount of episodes need to be performed. Training a large number of episodes on a real setup would be impractical because it would take too long, but the deterministic nature of the problem allow us to use simulation instead. This is because the policy can be trained without taking motion planning into account, since the state update rule does not depend on the motion plan. We only need to calculate the robot poses at the drop-off and pick-up locations. The trained policy in simulation will then be executed on the real setup. However, the accuracy of this translation from simulated to real environment depends on the accuracy of the models and the repeatability of the two robotic manipulators. The closed-loop behavior of the supervisory controller can help mitigate some of these issues.

C. Related work

A previous method presented by De Witte et al. [12] uses a discretized version of the workspace, instead of using a continuous parametrization. A policy is found using DQN [9], where the optimal action-value function is approximated using neural networks $Q(\mathbf{s}, \mathbf{a}; \theta) \approx Q^*(\mathbf{s}, \mathbf{a})$. The best action is then found by taking the action with maximum value $\pi(\mathbf{s}) = \max_a Q(s, a; \phi)$. Discretizing this shared workspace results in a high-dimensional solution space that is hard to solve, resulting in longer training times.

Additionally, an extensive reward shaping and extensive and complex pretraining, with multiple networks, are required for architecture B. Architecture A on the other hand required two policies, due to how the information of the goal configuration was implemented. Moreover, the policy is only trained to output a certain set of discrete possible positions. If this is applied on a real set-up, where the execution of actions is no longer deterministic, the actual achieved position may differ, resulting in reduced accuracy of the goal task, as this position is not yet seen before by the policy. All of these problems are mostly mitigated in our approach.

IV. Experimental Validation

A. Set-up

The set-up uses two Lynxmotion AL5A robotic arms with five degrees of freedom to perform pick-and-place maneuvers on a square object by grasping a toothpick attached at its center. The robots are controlled in an open-loop manner, moving from the pick-up position to the drop-off position. An Intel RealSense Depth Camera D435 is used to perceive information from the system, such as the configuration of the block and the goal.

The current path planning method uses a parametric trajectory optimization. The trajectory ζ is parametrized in joint coordinates using interpolating B-splines. The number of knots m and the degree d determine the number of splines, $N = m + d - 1$. In our experiments (m, d) are chosen as $(3, 5)$. As the problem is purely kinematic and dynamics are ignored, time parametrization is of no importance and we parametrize time with an arbitrary variable $\tau \in [0, 1]$.

$$\zeta(\tau; \theta) = \sum_{i=1}^{N} B_{i,d}(\tau)\theta_i$$

Collision avoidance is considered for both the ground and the robots themselves, with the robots being limited in the mutual workspace to prevent collisions with each other. The position in Cartesian coordinates of all the links is evaluated using 10^2 intervals. The optimization minimizes the distance in Cartesian coordinates, with the integral being calculated with trapezoidal integration.

The path planning can be implemented using any existing trajectory optimization or motion planning method, it is not limited to the one used in our approach, due to the decoupling of the task and motion planning.

B. Implementation

All code is written in Python. The reinforcement learning algorithms were implemented using the stable baselines 3 package [18].

1) Continuous approximation: A two-dimensional grid of points at a fixed height, spaced apart 5 mm in each direction, is sampled. For each point the feasibility is checked for both robots, resulting in a discrete approximation of the manipulation spaces. Taking the intersection results in a discrete approximation of the mutual manipulation space, which can be used to find the continuous approximation. The resulting $\underline{r}$ and $\overline{r}$, are 150 and 230 respectively. The goal and initial configuration positions are sampled from the union of the two discrete manipulation spaces. These discrete approximations, together with the continuous approximation in black, are given in Fig. 4. The base of both robots are indicated as well.

2) Hyperparameter tuning: An extensive hyperparameter search is performed for all different architectures using Optuna [19]. The resulting hyperparameters for all continuous cases can be found in table I. ReLU is used as activation function and Adam [20] as optimizer. Architecture B is trained using HER, with five goals being generated from all states of an episode. For the first architecture HER seems

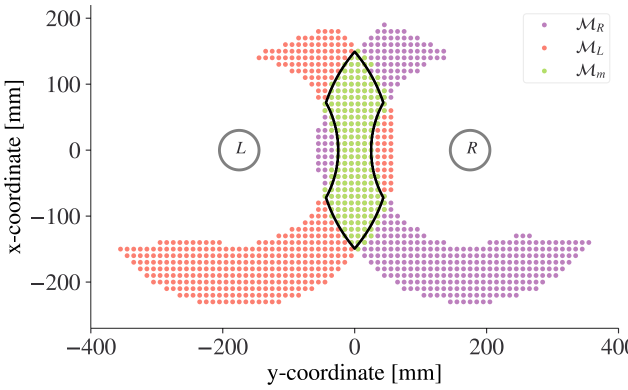


Fig. 4: Continuous approximation shown on top of discrete approximations.

to provide no added benefit, which is to be expected. The maximum allowed episode length is twenty, while taking a random policy results in an average length of around 10. This means that using HER results in completed episodes of length twenty being added to the replay buffer, which is rather long in comparison to a random policy. When limiting this maximum episode length to e.g. five, HER could help. The second architecture clearly benefits from using HER, on which we will elaborate in the results section. Additionally, since the final step is not dictated by the policy, the goals can only be generated from the final state of the episode, while for architecture B it is possible to generate the goals from all states in the episode.

TABLE I: Hyperparameters SAC for both architectures.

Hyperparameter	Architecture A	Architecture B
learning rate	0.0003	0.0005
discount (γ)	0.99	0.99
replay buffer size	10^5	10^5
batch size	32	128
learning starts	0	1000
train frequency	32	16
target smoothing coefficient (τ)	0.01	0.01
number of hidden layers	2	3
number of hidden units per layer	400|300	512

C. Results

The results are compared to the method presented in III-C. The main improvement can be found in the ease of training and the higher success rate for architecture B.

1) Training: Training is performed with the optimal hyperparameters mentioned previously.

A first comparison is made between the simplified form of curriculum learning and learning directly in the final environment. It can be seen, that while training seems to move towards an optimal point, it does so slower than the simplified form. This gives a validation for the use of the curriculum learning, as training speed is increased. On the other hand, it also shows that curriculum learning is not required and a policy can be found without it. Here, the first clear benefit of the method over the discretized method surfaces, training is much simpler. On top of being simpler,

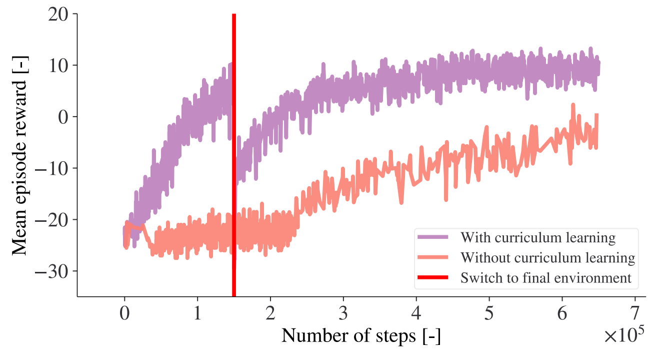


Fig. 5: A comparison made between curriculum learning and no curriculum learning, the switch of environments for curriculum learning is indicated at step 150,000.

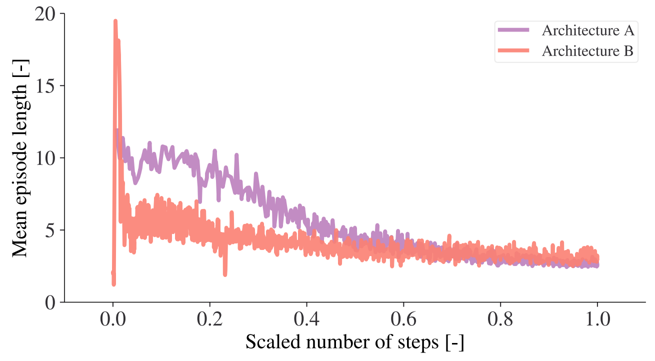


Fig. 6: A comparison made of the mean episode length during training for both architectures. Architecture A is trained for 250,000, while architecture B is trained for 650,000. The number of steps are scaled for a better comparison.

it is significantly faster for architecture B (2,150,000 vs 650,000 steps).

Secondly we can take a look at the mean episode length during training for both methods. The main difference is the use of HER, training architecture B is done using HER, which can be seen by the sudden decrease in mean episode length. As shorter episodes that reach their goal are created. Additionally, since the policy dictates when an episode is done, even a bad policy will result in shorter episodes.

It can be noted that uniform sampling of α and β will result in a non-uniform distribution of points in Cartesian coordinates, because of the changing width. While this seems suboptimal at first sight, the non-uniform sampling may benefit the exploration phase. The positions with a higher density lie on the edges of the mutual manipulation space. Switches between edges results in a higher change of relative orientation, thus benefitting the exploration.

2) Validation: We perform a number of experiments with a randomly sampled goal and start state for all four architectures, both in simulation as on the set-up. The average episode length and the success rate are given. The success rate is given as the percentage of episodes that resulted in a desired goal; 50 episodes are performed.

The results in simulation and on the setup are given in Fig.

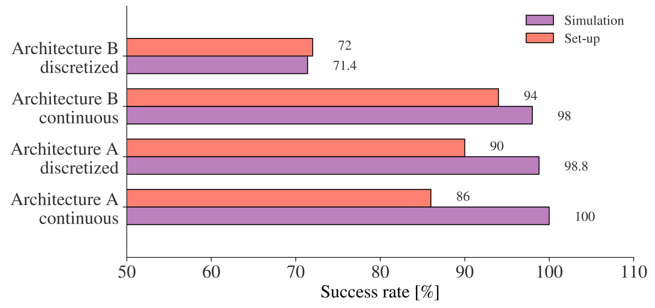


Fig. 7: A comparison made between the discretized version and the continuous version of the shared workspace, both for architecture A and B. The success ratios are shown both on the set-up as in simulation.

7 and 8. A comparison is made with the discretized version of the shared workspace for both architectures. The first figure shows the success rate, while the second shows the average number of steps in an episode. A clear improvement can be seen when using the continuous representation over the discretized, especially for architecture B, both on the setup as in simulation and both in episode length as in success rate. While the same can be said for architecture A and its episode length, the success rate is more or less the same. It shows that for the simpler problem statement this method doesn't provide a massive boost in performance, while for the harder problem statement the parametrization of the mutual workspace using continuous variables is almost required.

For training in simulation no physics engine was required, since only the initial x_t and final state x_{t+1} of the motion plan where needed. However for the evaluation, a physics engine is used to be as complete as possible. The robots' dynamics and interaction with the environment are simulated using PyBullet. By modeling the robot in a physics engine, we are able to better validate the trained policies.

For architecture B, the possible end state are: (a) Correct position and orientation, (b) correct position and wrong orientation, (c) wrong position. The wrong position can be achieved in two ways, if the wrong robot is asked to perform the final step it could be possible the position lies outside the manipulation space of that robot, or if the max amount of steps is achieved. Of the 50 performed episodes two resulted in the final state (b) and one in (c) by querying the wrong robot. For architecture A, all episodes where terminated before maximum number of steps, so the low success rate is due to wrong model of the robot. The latter is less of an issue for architecture B, since the model is not used to dictate the final step, resulting in a better translation to the set-up.

V. CONCLUSIONS

This work presents a method for enabling multi-robot manipulation in the presence of under-actuation by utilizing the shared workspace. The proposed approach involves decomposing the resulting Task and Motion Planning problem and using a supervisory control system to output the next best action in real-time. The next best action is selected from a continuous parametrization of the shared workspace by a

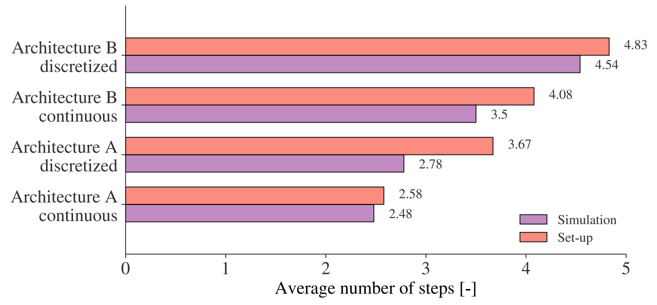


Fig. 8: A comparison made between the discretized version and the continuous version of the shared workspace, both for architecture A and B. The average number of steps are shown both on the set-up as in simulation.

parametrized policy. This continuous approximation results in a simple to train policy, using only sparse rewards, with a shorter training time and a higher success rate than previous methods. A translation to a real set-up shows it's application in the real-world.

The parametrization of actions into continuous variables can be extended to other problems as well, enabling us to find an action sequence by learning a policy. An other example could be to parametrize the workspace of a single robot, and combine this with a number of individual actions, which could be parametrized in a similar way. Cooperation could also be performed by parametrization of a three-dimensional workspace, making it for example possible to perform hand-over actions.

In future work, it may be possible to extend the system to include more robots or a greater range of possible actions, and to consider different types of goal configurations. Another step could be to implement the method with a combined Task and Motion Planning solver, resulting in a faster combined optimization. Additionally, it could be possible to give the policy more information of its environment to make it even more contextual. A variable representing obstacles could be included in the augmented state, such that the policy can redirect its next position.

REFERENCES

[1] S. Kim, H. Seo, J. Shin, and H. J. Kim, "Cooperative aerial manipulation using multirotors with multi-dof robotic arms," IEEE/ASME Transactions on Mechatronics, vol. 23, no. 2, pp. 702–713, 2018.

[2] Q. M. Ta and C. C. Cheah, "Multi-agent control for stochastic optical manipulation systems," IEEE/ASME Transactions on Mechatronics, vol. 25, no. 4, pp. 1971–1979, 2020.

[3] C. Viegas, M. Tavakoli, P. Lopes, R. Dessi, and A. T. de Almeida, "Scala—a scalable rail-based multirobot system for large space automation: Design and development," IEEE/ASME Transactions on Mechatronics, vol. 22, no. 5, pp. 2208–2217, 2017.

[4] M. Dogar, A. Spielberg, S. Baker, and D. Rus, "Multi-robot grasp planning for sequential assembly operations," in IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 193–200.

[5] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," Annual Review of Control, Robotics, and Autonomous Systems, vol. 4, no. 1, pp. 265–293, 2021. [Online]. Available: https://doi.org/10.1146/annurev-control-091420-084139

[6] D. Driess, J.-S. Ha, and M. Toussaint, "Deep Visual Reasoning: Learning to Predict Action Sequences for Task and Motion Planning from an Initial Scene Image," in Proceedings of Robotics: Science and Systems, Corvalis, Oregon, USA, July 2020.

[7] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel, "Guided search for task and motion plans using learned heuristics," in 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 447–454.

[8] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering atari, go, chess and shogi by planning with a learned model," Nature, vol. 588, no. 7839, pp. 604–609, Dec. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-03051-4

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[10] H. Nguyen and H. La, "Review of deep reinforcement learning for robot manipulation," in Third IEEE International Conference on Robotic Computing (IRC), 2019, pp. 590–595.

[11] A. Singh, L. Yang, C. Finn, and S. Levine, "End-to-end robotic reinforcement learning without reward engineering," in Proceedings of Robotics: Science and Systems, FreiburgimBreisgau, Germany, June 2019.

[12] S. De Witte, T. Van Hauwermeiren, T. Lefebvre, and G. Crevecoeur, "Learning to cooperate: A hierarchical cooperative dual robot arm approach for underactuated pick-and-placing," IEEE/ASME Transactions on Mechatronics, vol. 27, no. 4, pp. 1964–1972, 2022.

[13] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 1582–1591. [Online]. Available: http://proceedings.mlr.press/v80/fujimoto18a.html

[14] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1506.02438

[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in International Conference on Machine Learning, 2018.

[16] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," in Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf

[17] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in Proceedings of the 26th Annual International Conference on Machine Learning, ser. ICML '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 41–48. [Online]. Available: https://doi.org/10.1145/1553374.1553380

[18] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," Journal of Machine Learning Research, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[19] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980