Learning Manipulation Tasks: A multi-agent approach

Technical Report No. CCC-23-004

by

M.Sc. Jorge Eduardo Gutiérrez Gómez

Doctoral Advisors:

Dr. Luis Enrique Sucar Succar, INAOE

Dr. Rafael Eric Murrieta Cid, CIMAT

Instituto Nacional de Astrofísica, Óptica y Electrónica

©Coordinación de Ciencias Computacionales

January, 2023

Luis Enrique Erro 1

Santa María de Tonantzintla,

Puebla, CP 72840

Contents

Abstract

The main topic of robotic manipulation is to modify the state of objects with a robot, mostly robotic arms are used.

The objective of this project is to create a system capable of interacting with several home objects, such as glasses, cups, cans, etc. So that the arm can be useful in human activities on a daily basis.

The way this problem is addressed is by segmenting the robot itself. Creating a modular system. The robot is segmented into its most relevant parts, for example, the gripper, the arm, and the robotic base. For each segment an agent is designed to control it. To achieve this, several stages of machine learning training are used, each of the segments is trained independently until the segment's own objectives are achieved. Then a task learning system is designed based on a learning system by demonstration and imitation using videos of a person doing the activities. Ultimately the entire system will perform manipulation tasks that are useful to humans.

Keywords— Robotic manipulation, Computer sciences, Multi-agent

1 Introduction

Robotic manipulation is a central axis in robotics for its ability to change the world around it. The potential of robotic manipulation is immense. It as has been demonstrated in the industry where environments are well structured and controlled. Recent advances in algorithms and perception, allow its use in less controlled and humman common environments, such as a kitchen for food preparation, operating rooms, nuclear plants, also in some of the most fundamental and necessary activities like agriculture. These complex environments require increasingly intelligent and dynamic behaviors that overcome the challenges posed by the environment [1, 2, 3, 4].

Researchers have focused on trying to answer questions like. How could a robot identify and learn to manipulate objects around it? How could a robot avoid hitting objects between it and the objective? How could a robot learn faster and better? The research ranges from learning concrete skills through human demonstration, to learning abstract descriptions of a manipulation task for high-level planning, to discovering the functionality of an object in order to interact with it, among others. Robotic manipulation must address grasping applications that go beyond picking up and putting down an object. Skills motivated by the human ability to manipulate its environment. Robotic manipulation is commonly configured in such a way that in the final effector of a manipulator robot, are grippers or robotic hands in different configurations used for such purposes, which gives versatility to the task of grasping objects [5, 6, 7, 8, 9].

Related to this work, same as others works we aim to develop a robotic system capable of dexterous manipulate common objects, our approach similar to other works relays its attention in two main ideas, *modular learning, and the development of control and coordination algorithms for a multi-agent system* [10, 11, 12, 13, 14, 15, 16, 17]. In contrast to these works we aim to generalize the learning between agents, that is we try to train every agent in the same way, with the same kind of information.

With this approach we aim to develop a learning system that can train a robotic arm part by part, for example train the gripper and the arm as different agents over two different stages, for later use a coordination algorithm to move the robotic arm as a whole to achieve dexterous manipulation. While a modular learning divide a large task into different small ones, this helps to focus the train in the stages of the task that can be more difficult, using different methods according to the task. If we consider that a task is a sequence of several other small task we can first learn those small task and then concatenate them to do the main task.

1.1 Motivation

The development of robotic manipulation is motivated by the human vision of the potential of robotics in daily life. In controlled environments like the industry, robotic manipulators have proven to be extremely useful when performing repetitive tasks that require a very high degree of precision. Although the human desire to have robots in daily life arose many years ago, recently due to the increasing industrialization and cheaper sensors, actuators, and rapid prototyping systems, they provide the right environment for the development of cheaper and more accessible robots. These new robotic systems require to face challenging, dynamic environments with a high degree of uncertainty. These environments demand flexible robots in terms of their ability to adapt, both physically and in their behavior, with sensors, actuators, and algorithms capable of translating this information into useful behaviors, in order to be able to interact with the surrounding environment. Regarding the approach of computational sciences for robotic manipulation, the algorithmic approach that leads to the development of adequate behaviors to manipulate objects and interact with the environment is of greater interest [1, 2, 3, 4].

In the state of the art there are several approaches to deal with robotic manipulation, recently most of them focus in the use of neural networks using reinforcement learning [5, 6, 7, 8, 9], this approach has the capacity of learn some basic movements using the whole arm. In contrast to our approach, in which each section of the robot will be trained separately to speed up learning. Each trained section may be used in various combinations, with other sections, for example, exchanging grippers. The coordination system is expected to handle these changes as well, reusing previously acquired knowledge. Changes in the morphology of the robot as a whole will then not require retraining.

We also consider that divide tasks is a more efficient way to perform complex tasks that include many steps. The first focal point task that is recurrent in robotic handling is to bring the gripper closer to a certain position and grab an object, and from which other tasks can be performed for example, motion planning, obstacle avoidance, complex task learning and so on [10, 11, 12, 13, 14, 15, 16, 17]

1.2 Justification

The development of algorithms and learning methodologies that provide robots with dexterous manipulation of objects around them is a central research axis in robotics. The ability of a robot to manipulate its environment depends directly on its behavioral policies. Same as must be developed very carefully, taking into account its physical aspects, such as number of joints, power of its motors, workspace that can be reached, sensors and information available to it. Also considering physical aspects of the objects to be manipulated, whose information is very useful to adequately perform the tasks for which it was programmed.

In a home environment, a robot that can dynamically manipulate its environment is extremely useful, even if the robot may not be as precise at manipulation. For example, a common problem in robotic manipulation research is to endow the robot with of the ability to take objects and give them to a person, in this way it could help people with limited mobility in their daily activities, passing objects that may be useful and difficult to reach, such as medicines, bottles of water, even the TV controller, a cell phone, among other common objects. These manipulation activities can be useful in other contexts, such as helping with cleaning, picking up objects and organizing them, etc.

It's propose that the tasks be divided according to the agent that performs them. The basic tasks learned by the agents will then be homologate to be used in conjunction with the other agents.

Regarding the multi-agent approach, some authors have work on this kind of approach, using centralized and decentralized schemes, with partial or no communication [11]. Most of them used more than one robot, to show the coordination and cooperation scheme.

The approach proposed in this work is to use agents in fundamental sections of one single robot, for example the base of the robot, the body of the arm, and the gripper. then a coordination agent system will take the agents and make them cooperate to achieve the task. This approach will allow the robot to reuse certain components and still be functional, it will also allow it to be trained by parts, and even make changes in its morphology without being affected, for example, if the gripper is changed the arm will still be able to reach the objects proposed by the system, or the arm can be mounted on different robotic platforms.

The advantages of using a multi-agent approach over a single-agent, is a faster learning because the sub-systems will be less complex than the whole system, it will also be easier to identify where the robot is failing without the need to include the entire system. By working with more simple systems, experiments can be repeated without much time penalty, and by dividing the robot into different modules that learn separately it even allows one to exchange sections of the robot and still be operational. Regarding the methodology of first using a 2D simulator and then a 3D simulator to finally implement it in reality, it allows us to test multiple algorithms in simple environments faster than a very complex implementation on a real robot, it also allows us to do developments without risking the physical integrity of the laboratory equipment. Some of the disadvantages of using this methodology could be that the algorithms developed in 2D or in the same 3D simulator are not fully usable in a real environment, it could also be that an error in an agent propagates an error throughout the system, or that the coordination system is not sufficient to control the systems coherently.

This research on robotic manipulation is feasible, because the necessary resources are available to be carried out, such as robotic platforms, computers, and computer peripherals.

Regarding ethical considerations, this project aims to develop algorithms and methodologies that allow increasing the state of the art of robotic manipulation by exploring different ideas and methodologies. This project is focused on the expansion of the application of algorithmic knowledge for the human development, it is not intended at any time that the research be oriented towards robots harming people or living beings that surround them with whom they could interact.

1.3 Problem Statement

Robotic manipulation consists of using robots to change the state of the objects around it. Robotic arms with end effectors or grippers suitable for these tasks are commonly used, some of the most used being those that are made up of fingers, although there are other configurations, for example, robots with suction cups or grippers.

Most stages that make up robotic manipulation are challenging, from the fact of being able to identify the object to be manipulated is a challenge to even understanding how to hold it and later generate the motor skills to do it, understanding the tasks and their implications are also problems to consider.

There are several huge problems relate to robotic manipulation. This work focus it's efforts in motion planning, which is the way in which the robotic system plans and executes its movements. The tools available and which will be used are a Kinova robotic arm, which has the ability to be commanded by ROS to control the position of its joints, both in position and speed. It also has the feature to feedback the current exerted by its motors, this arm is compatible with gazebo simulator.

In the state of the art there are several approaches to motion planning in robotic manipulation [18, 19], some of the most recent ones address the problem using neural networks and machine learning, for example [20, 21, 22, 23, 24, 25].

In general, the idea is based on training a neural network using a Markov Decision Process (MDP) [26, 27] with the states, goals and actions that can be carried out by the robot.

Following this methodology, the neural network process the current states of the robot and the target and returns a value corresponding to the movement that the robot must perform. Either as a value of speed, or position of the joints, or of the end effector.

The biggest problems for this methodology correspond to the representation and treatment of the data of the states of the system, that can contain all the necessary information so all the agents could be coordinated.

Other problem corresponds to the number of samples necessaries for the correct training. The number of joints exponentially increases the complexity of the problem due to the number of possible states. Although in continuous states the network can infer actions from states not seen before, it requires a large number of samples. Some authors have suggested that the system should explore the state space automatically and stochastically. This kind of exploration suffers from major problems such as falling into local maximums, and an exploration that takes too long.

In reinforcement learning that uses neural networks for robotic manipulation, several authors agree that the training time required to perform manipulation is very long, requiring thousands of training episodes. Small modifications of the robot's morphology often require a retrain of the whole system. Due to the large amount of time required for training in robotic manipulation, especially with physical robots, most authors prefer to use simulators and later transfer the training to a real robot, but in all cases, the robot being trained in simulation is the same as the physical one.

Some authors have draw this problem and accelerated learning by giving demonstrations to the training system of how the task should be done, through videos or simulations as is shown in [5, 6], some others have used some novel reinforcement learning algorithms [5, 9], which greatly improves exploration but are still expensive in number of steps.

This thesis focus on motion planning, and the amount of learning time problem, generality of the algorithms and modularity of the systems, through a multi-agent approach.

The multi-agent approach intends to assign different tasks to segments of a robot using different agents, which will act together to solve the tasks, because it is easier to train agents that have less space for actions, due to the number of possible combinations that can be generated.

The problem of robotic manipulation with cooperative controllers has been studied from a centralized point of view, where one agent controls the behavior of other agents, and also in a decentralized way, where each agent determines its own actions, with partial or no communication [6].

Several of these works focus on using various cooperative robots to accomplish manipulation tasks, [6]. In our case we approached the problem in a single robot, there are works that have dealt with manipulation on a single robotic arm but have not achieved good results regarding joint learning [10].

The main challenges and problems of this multi-agent approach to robotic manipulation and that will be addressed in this work are the following: Properly finding to what extent a robot can be segmented into different agents and still achieve manipulation activities.

Design a coordination system that is not more complex than training the robot without segmenting the robot. Finding an adequate representation of the robot itself and its environment, with its action space so that learning and training can be easily replicated between different robots with similar characteristics.

1.4 Research Questions

Can a manipulator robot, segmented and distributed among different agents, learn in a generalized way, faster and better, using reinforced learning, than a manipulator robot made up of a single agent, with the same reinforcement learning technique?

To what extent can a robot be segmented and distributed among different agents and still achieve manipulation activities?

Is it possible to design a coordination system between agents that is simple enough to be easier than training a centralized robot?

What is the best representation of a manipulator robot, its environment and space of actions, to use machine learning, that can be replicated between different robots?

1.5 Hypothesis

Training a robot for robotic manipulation through reinforcement learning, with precise and adequate segmentation, requires fewer steps and is more generalizable compared to training through reinforcement learning of a single agent system.

1.6 Objectives

1.6.1 General objective

- Develop a machine learning and agent coordination system for robotic manipulation of complex tasks.

1.6.2 Specific objectives

- Design a simple 2D simulator so that different segments of a robot can be simulated separately and together. In order to implement some of the most successful reinforcement machine learning algorithms in the state of the art over a segmented and centralized robot.
- Using the agents trained for the different segments of the robot and develop a distributed coordination system that handle the several agents for manipulation tasks.
- Consider conflicts between agents and compare the RL training of separate agents with coordination against the training as a whole.
- Design a training and coordination system based on the experiments and algorithms treated with the 2D simulator, in a 3D simulator based on the Kinova Jaco robotic arm. Using three agents, robotic base, arm, and gripper.

- Adapt the systems developed in the 2D and 3D simulator so that they can be used in a real Kinova-Jaco robot for manipulation tasks.
- Evaluate the manipulation behaviors arising from the distributed learning and coordination system, from motor control. And compare them with a centralized manipulation system.

1.7 Scope and Limitations

The main limitations to achieve a precise manipulation is the perception of the environment and the proprioception of the robot itself. Even if proper training has been carried out, the manipulation may fail if the robot itself is out of calibration or cannot clearly identify objects in the environment. Because the main focus is on multi-agent learning, a controlled environment will be used to perform the manipulation activities.

Robotic manipulation is understood to be the ability of a robotic platform to move objects in space, changing their absolute position and orientation, for whatever purpose.

The focus is on the manipulation of solid objects, mainly those commonly used in the home.

The robotic manipulation behaviors that are intended to be achieved are: approaching the target object, taking it in an appropriate way, understanding a firm grip that does not compromise the physical integrity of the object or the robot. This subjection should preferably be carried out as a human being would. Safely place the object on a surface designated for that purpose. Push objects, pull objects, turn and follow trajectories with them. By pushing and pulling objects it is about directing objects with a predefined trajectory. From one point to another.

1.8 Expected Contributions

The main contribution of this work is found in the development of a multi-agent distributed system that will reduce the training time for robotic manipulation systems compared to centralized systems.

This system will be able to reduce the training time through two techniques, the first is to segment the robot, design an agent for each one and train with specialized tasks for each segment, for example, a robotic arm without a gripper needs training to approach objects without crashing, but it does not require specializing in holding each one of them. On the other hand, a gripper requires training to hold different morphologies of objects, but does not require training in motor control to approach them.

The second is related to the ability to transfer learned skills from one purpose to another and between different tasks. For example, the robotic arm that has learned to approach an object without colliding, can approach another object without the need for retraining. In this case, the precise grip training falls on the gripper agent. Following this methodology, each agent can be trained to perform certain repetitive tasks without the need to train the robot as a whole.

2 Background

Machine learning: Learning is the main hallmark of human intelligence and the basic means to obtain knowledge. Machine learning is a subject that studies how to use computers to simulate human learning activities, and to study self-improvement methods of computers to obtain new knowledge and new skills. Knowledge discovery is a process to identify effective, novel, potential, useful and understanding model from large amounts of data. [28]

Artificial neural networks (ANN): An artificial neural network or simply neural network consist of an input layer of neurons or nodes, one or two or even more hidden layers of neurons, and a final layer of output neurons. Figure 1 shows a typical architecture, where the lines connecting neurons are also shown. Each connection is associated with a numeric number called weight. The output h_i of neuron i in the hidden layer is,

$$h_i = \sigma(\sum_{j=1}^{N} V_{ij}x_j + T_i^{hid})$$

Where $\sigma()$ is called activation or transfer function, N the number of input neurons, V_{ij} the weights, x_j inputs to the input neurons, and T_i^{hid} the threshold terms of the hidden neurons. [29]

Convolutional neural networks (CNN): Introduced by Le Cun, are a class of biologically inspired neural networks which solve

$$\widehat{f} = argmin_{f \in F} E[L(Y, f(X))]$$

By passing X through a series of convolutional filters and simple non-linearities. Figure 2 shows a typical CNN architecture. [30]

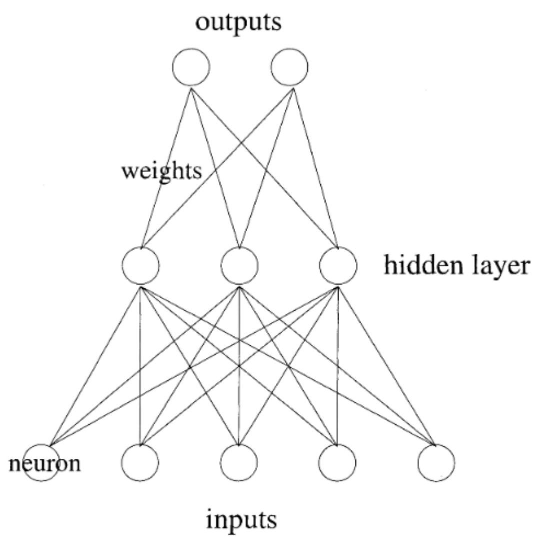


Figure 1: Architecture of a neural network, from [29]

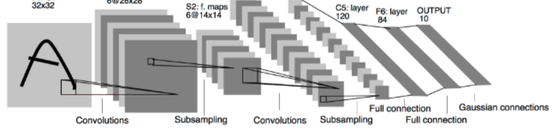


Figure 2: Architecture of a convolutional neural network, from [30]

2.1 Reinforcement Learning

This subsection was taking from [31].

The main characters of RL are the agent and the environment. The environment is the world that the agent lives in and interacts with. At every step of interaction, the agent sees a (possibly partial) observation of the state of the world, and then decides on an action to take. The environment changes when the agent acts on it, but may also change on its own.

The agent also perceives a reward signal from the environment, a number that tells it how good or bad the current world state is. The goal of the agent is to maximize its cumulative reward, called return. Reinforcement learning methods are ways that the agent can learn behaviors to achieve its goal.

Additional terminology is described below.

2.1.1 States and Observations

A state s is a complete description of the state of the world. There is no information about the world which is hidden from the state. An observation o is a partial description of a state, which may omit information.

In deep RL, we almost always represent states and observations by a real-valued vector, matrix, or higher-order tensor. For instance, a visual observation could be represented by the RGB matrix of its pixel values; the state of a robot might be represented by its joint angles and velocities.

When the agent is able to observe the complete state of the environment, we say that the environment is fully observed. When the agent can only see a partial observation, we say that the environment is partially observed.

2.1.2 Action Spaces

Different environments allow different kinds of actions. The set of all valid actions in a given environment is often called the action space. Some environments, like Atari and Go, have discrete action spaces, where only a finite number of moves are available to the agent. Other environments, like where the agent controls a robot in a physical world, have continuous action spaces. In continuous spaces, actions are real-valued vectors.

2.1.3 Policies

A policy is a rule used by an agent to decide what actions to take. It can be deterministic, in which case it is usually denoted by μ:

$$a_t = \mu(s_t)$$

or it may be stochastic, in which case it is usually denoted by π:

$$a_t \sim \pi(\cdot|s_t)$$

Because the policy is essentially the agent's brain, it's not uncommon to substitute the word "policy" for "agent", eg saying "The policy is trying to maximize reward."

In deep RL, we deal with parameterized policies: policies whose outputs are computable functions that depend on a set of parameters (eg the weights and biases of a neural network) which we can adjust to change the behavior via some optimization algorithm.

We often denote the parameters of such a policy by θ or ϕ, and then write this as a subscript on the policy symbol to highlight the connection:

$$a_t = \mu_\theta(s_t)$$
$$a_t \sim \pi_\theta(\cdot|s_t)$$

Stochastic Policies: The two most common kinds of stochastic policies in deep RL are categorical policies and diagonal Gaussian policies.

Categorical policies can be used in discrete action spaces, while diagonal Gaussian policies are used in continuous action spaces.

Two key computations are centrally important for using and training stochastic policies: Sampling actions from the policy and Computing log likelihoods of particular actions, log $\pi_\theta(a|s)$

In what follows, we'll describe how to do these for both categorical and diagonal Gaussian policies.

Categorical Policies A categorical policy is like a classifier over discrete actions. You build the neural network for a categorical policy the same way you would for a classifier: the input is the observation, followed by some number of layers (possibly convolutional or densely-connected, depending on the kind of input), and then you have one final linear layer that gives you logits for each action, followed by a softmax to convert the logits into probabilities.

Log-Likelihood. Denote the last layer of probabilities as $P_\theta(s)$. It is a vector with however many entries as there are actions, so we can treat the actions as indices for the vector. The log likelihood for an action a can then be obtained by indexing into the vector:

$$\log \pi_\theta(a|s) = \log \left[P_\theta(s)\right]_a$$

Diagonal Gaussian Policies A multivariate Gaussian distribution, is described by a mean vector, μ, and a covariance matrix, Σ. A diagonal Gaussian distribution is a special case where the covariance matrix only has entries on the diagonal. As a result, we can represent it by a vector.

A diagonal Gaussian policy always has a neural network that maps from observations to mean actions, $\mu_\theta(s)$. There are two different ways that the covariance matrix is typically represented.

The first way: There is a single vector of log standard deviations, $\log \sigma$, which is not a function of state: the $\log \sigma$ are standalone parameters.

The second way: There is a neural network that maps from states to log standard deviations, $\log \sigma_\theta(s)$. It may optionally share some layers with the mean network.

Note that in both cases we output log standard deviations instead of standard deviations directly. This is because log stds are free to take on any values in $(-\infty, \infty)$, while stds must be nonnegative.

Sampling. Given the mean action $\mu_\theta(s)$ and standard deviation $\sigma_\theta(s)$, and a vector z of noise from a spherical Gaussian ($z \sim \mathcal{N}(0, I)$), an action sample can be computed with

$$a = \mu_\theta(s) + \sigma_\theta(s) \odot z$$

Where $\odot$ denotes the elementwise product of two vectors.

Log-Likelihood. The log-likelihood of a k -dimensional action a, for a diagonal Gaussian with mean $\mu = \mu_\theta(s)$ and standard deviation $\sigma = \sigma_\theta(s)$, is given by

$$\log \pi_\theta(a|s) = -\frac{1}{2}\left(\sum_{i=1}^{k}\left(\frac{(a_i - \mu_i)^2}{\sigma_i^2} + 2\log\sigma_i\right) + k\log 2\pi\right)$$

2.1.4 Trajectories

A trajectory τ is a sequence of states and actions in the world,

$$\tau = (s_0, a_0, s_1, a_1, ...)$$

The very first state of the world, s_0, is randomly sampled from the start-state distribution, sometimes denoted by ρ_0:

$$s_0 \sim \rho_0(\cdot)$$

State transitions (what happens to the world between the state at time t, s_t, and the state at $t+1$, s_{t+1}), are governed by the natural laws of the environment, and depend on only the most recent action, a_t. They can be either deterministic,

$$s_{t+1} = f(s_t, a_t)$$

Or stochastic,

$$s_{t+1} \sim P(\cdot|s_t, a_t)$$

Actions come from an agent according to its policy.

Trajectories are also frequently called episodes or rollouts.

2.1.5 Reward and Return

The reward function R is critically important in reinforcement learning. It depends on the current state of the world, the action just taken, and the next state of the world:

$$r_t = R(s_t, a_t, s_{t+1})$$

Although frequently this is simplified to just a dependence on the current state, $r_t = R(s_t)$, or state-action pair $r_t = R(s_t, a_t)$.

The goal of the agent is to maximize some notion of cumulative reward over a trajectory, but this actually can mean a few things. We'll notate all of these cases with $R(\tau)$, and it will either be clear from context which case we mean, or it won't matter (because the same equations will apply to all cases).

One kind of return is the finite-horizon undiscounted return, which is just the sum of rewards obtained in a fixed window of steps:

$$R(\tau) = \sum_{t=0}^{T} r_t$$

Another kind of return is the infinite-horizon discounted return, which is the sum of all rewards ever obtained by the agent, but discounted by how far off in the future they're obtained. This formulation of reward includes a discount factor $\gamma \in (0, 1)$:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

An infinite-horizon sum of rewards may not converge to a finite value, and is hard to deal with in equations. But with a discount factor and under reasonable conditions, the infinite sum converges.

2.1.6 The RL Problem

Whatever the choice of return measure (whether infinite-horizon discounted, or finite-horizon undiscounted), and whatever the choice of policy, the goal in RL is to select a policy which maximizes expected return when the agent acts according to it.

To talk about expected return, we first have to talk about probability distributions over trajectories.

Let's suppose that both the environment transitions and the policy are stochastic. In this case, the probability of a T -step trajectory is:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$$

The expected return (for whichever measure), denoted by $J(\pi)$, is then:

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \underset{\tau\sim\pi}{E}[R(\tau)]$$

The central optimization problem in RL can then be expressed by

$$\pi^* = \arg\max_{\pi} J(\pi)$$

With π^* being the optimal policy.

2.1.7 Value functions

It's often useful to know the value of a state, or state-action pair. By value, we mean the expected return if you start in that state or state-action pair, and then act according to a particular policy forever after. Value functions are used, one way or another, in almost every RL algorithm.

There are four main functions.

- The On-Policy Value Function, $V^{\pi}(s)$, which gives the expected return if you start in state s and always act according to policy π:

$$V^{\pi}(s) = \underset{\tau\sim\pi}{E}[R(\tau)|s_0 = s]$$

- The On-Policy Action-Value Function, $Q^{\pi}(s,a)$, which gives the expected return if you start in state s, take an arbitrary action a (which may not have come from the policy), and then forever after act according to policy π:

$$Q^{\pi}(s,a) = \underset{\tau\sim\pi}{E}[R(\tau)|s_0 = s, a_0 = a]$$

- The Optimal Value Function, $V^*(s)$, which gives the expected return if you start in state s and always act according to the optimal policy in the environment:

$$V^*(s) = \max_{\pi} \underset{\tau\sim\pi}{E}[R(\tau)|s_0 = s]$$

- The Optimal Action-Value Function, $Q^*(s,a)$, which gives the expected return if you start in state s, take an arbitrary action a, and then forever after act according to the optimal policy in the environment:

$$Q^*(s,a) = \max_{\pi} \underset{\tau\sim\pi}{E}[R(\tau)|s_0 = s, a_0 = a]$$

There are two key connections between the value function and the action-value function that come up pretty often:

$$V^\pi(s) = \underset{a\sim\pi}{E} Q^\pi(s,a)$$

and

$$V^*(s) = \max_{a} Q^*(s,a)$$

2.1.8 The Optimal Q-Function and the Optimal Action

There is an important connection between the optimal action-value function $Q^*(s,a)$ and the action selected by the optimal policy. By definition, $Q^*(s,a)$ gives the expected return for starting in state s, taking (arbitrary) action a, and then acting according to the optimal policy forever after.

The optimal policy in s will select whichever action maximizes the expected return from starting in s. As a result, if we have Q^*, we can directly obtain the optimal action, $a^*(s)$, via

$$a^*(s) = \arg\max_a Q^*(s,a)$$

Note: there may be multiple actions which maximize $Q^*(s,a)$, in which case, all of them are optimal, and the optimal policy may randomly select any of them. But there is always an optimal policy which deterministically selects an action.

2.1.9 Bellman Equations

All four of the value functions obey special self-consistency equations called Bellman equations. The basic idea behind the Bellman equations is this:

The value of your starting point is the reward you expect to get from being there, plus the value of wherever you land next.

The Bellman equations for the on-policy value functions are

$$V^\pi(s) = \underset{\substack{a'\sim\pi \\ s'\sim P}}{E}[r(s,a) + \gamma V^\pi(s')]$$

and

$$Q^\pi(s,a) = \underset{s'\sim P}{E}[r(s,a) + \gamma \underset{a'\sim\pi}{E}[Q^\pi(s',a')]]$$

Where $s' \sim P$ is shorthand for $s' \sim P(\cdot|s,a)$, indicating that the next state s' is sampled from the environment's transition rules; $a \sim \pi$ is shorthand for $a \sim \pi(\cdot|s)$; and $a' \sim \pi$ is shorthand for $a' \sim \pi(\cdot|s')$.

The Bellman equations for the optimal value functions are

$$V^*(s) = \max_a \underset{s' \sim P}{E}[r(s,a) + \gamma V^*(s')]$$

and

$$Q^\pi(s,a) = \underset{s' \sim P}{E}[r(s,a) + \gamma \max_{a'} Q^*(s',a')]$$

The crucial difference between the Bellman equations for the on-policy value functions and the optimal value functions, is the absence or presence of the max over actions. Its inclusion reflects the fact that whenever the agent gets to choose its action, in order to act optimally, it has to pick whichever action leads to the highest value.

2.1.10 Advantage Functions

Sometimes in RL, we don't need to describe how good an action is in an absolute sense, but only how much better it is than others on average. That is to say, we want to know the relative advantage of that action. We make this concept precise with the advantage function.

The advantage function $A^\pi(s,a)$ corresponding to a policy π describes how much better it is to take a specific action a in state s, over randomly selecting an action according to $\pi(\cdot|s)$, assuming you act according to π forever after. Mathematically, the advantage function is defined by

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

2.1.11 Markov Decision Processes

Markov Decision Processes (MDP): Thomas [32] define a MDP as it provide a mathematical framework in which to study discrete-time decision-making problems. Formally, a Markov decision process is defined by a 7 tuple $(S, A, \mu_o, T, r, \gamma, H)$, where

- S is the state space, which contains all possible states the system may be in.
- A is the action space, which contains all possible actions the agent may take when interacting with the system.
- $\mu_o \in \delta(S)$ is the initial state distribution, a probability distribution over states in which the system will be initialized.
- $T : S \times A \longrightarrow \delta(S)$ is the transition dynamics. For each state s and action $a, T(s, a)$ yields a probability distribution over states that the system may transition into when taking action a from state s.
- $r : S \times A \times S \longrightarrow \mathbb{R}$ is the reward function. The value $r(s, a, s')$ gives the amount of "reward" associated transitioning into state s' when taking action a from state s.
- $\gamma \in [0, 1]$ is the discount factor, which determines how much future rewards should be "discounted" when making decisions. A value of $\gamma = 0$ means that we don't care about future rewards at all, while a value of $\gamma = 1$ indicates that rewards in the distant future should count just as much as the reward at the next time-step.
- H is the horizon, the maximum possible number of time-steps in each episode. It may be a positive integer (the finite-horizon case) or ∞ (the infinite-horizon case).

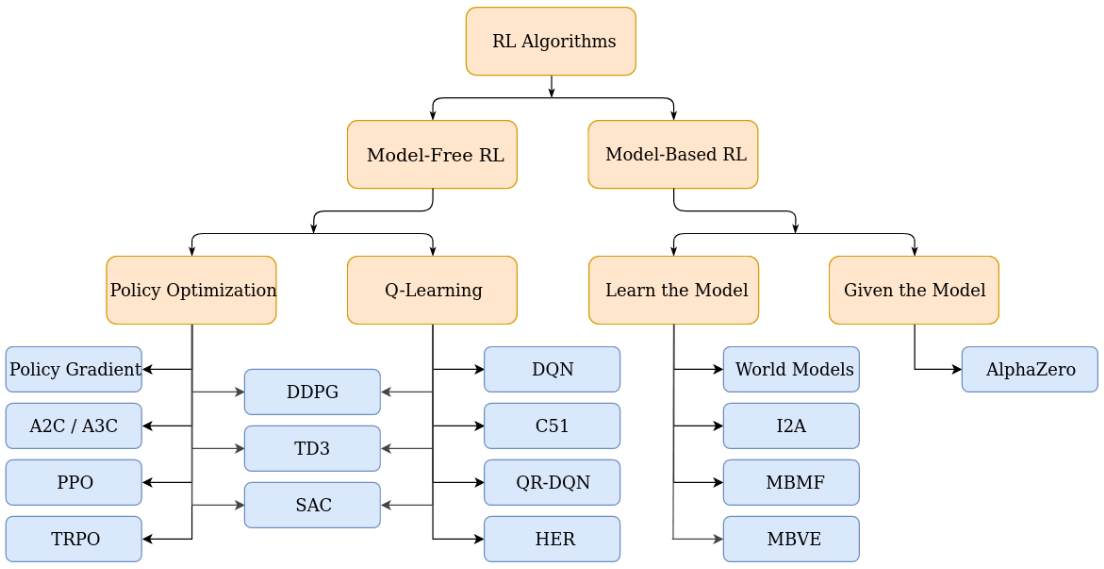


Figure 3: RL Algorithms

The name Markov Decision Process refers to the fact that the system obeys the Markov property: transitions only depend on the most recent state and action, and no prior history.

2.2 Kinds of RL Algorithms

Now that we've gone through the basics of RL terminology and notation, we can cover a little bit of the richer material: the landscape of algorithms in modern RL, and a description of the kinds of trade-offs that go into algorithm design.

2.2.1 Taxonomy of RL Algorithms

A non-exhaustive, but useful taxonomy of algorithms in modern RL 3.

- to highlight the most foundational design choices in deep RL algorithms about what to learn and how to learn it,

- to expose the trade-offs in those choices,
- and to place a few prominent modern algorithms into context with respect to those choices.

2.2.2 Model-Free vs Model-Base RL

One of the most important branching points in an RL algorithm is the question of whether the agent has access to (or learns) a model of the environment. By a model of the environment, we mean a function which predicts state transitions and rewards.

The main upside to having a model is that it allows the agent to plan by thinking ahead, seeing what would happen for a range of possible choices, and explicitly deciding between its options. Agents can then distill the results from planning ahead into a learned policy.

The main downside is that a ground-truth model of the environment is usually not available to the agent. If an agent wants to use a model in this case, it has to learn the model purely from experience, which creates several challenges. The biggest challenge is that bias in the model can be exploited by the agent.

Algorithms which use a model are called model-based methods, and those that don't are called model-free. While model-free methods forego the potential gains in sample efficiency from using a model, they tend to be easier to implement and tune.

2.2.3 What to Learn

Another critical branching point in an RL algorithm is the question of what to learn. The list of usual suspects includes

- policies, either stochastic or deterministic,

- action-value functions (Q-functions),
- value functions,
- and/or environment models.

What to Learn Model-Free There are two main approaches to representing and training agents with model-free RL:

Policy Optimization. Methods in this family represent a policy explicitly as $\pi_\theta(a|s)$. They optimize the parameters θ either directly by gradient ascent on the performance objective $J(\pi_\theta)$, or indirectly, by maximizing local approximations of $J(\pi_\theta)$. This optimization is almost always performed on-policy, which means that each update only uses data collected while acting according to the most recent version of the policy. Policy optimization also usually involves learning an approximator $V_\phi(s)$ for the on-policy value function $V^\pi(s)$, which gets used in figuring out how to update the policy.

A couple of examples of policy optimization methods are:

- A2C / A3C, which performs gradient ascent to directly maximize performance,
- and PPO, whose updates indirectly maximize performance, by instead maximizing a surrogate objective function which gives a conservative estimate for how much $J(\pi_\theta)$ will change as a result of the update.

Q-Learning. Methods in this family learn an approximator $Q_\theta(s, a)$ for the optimal action-value function, $Q^*(s, a)$. Typically they use an objective function based on the Bellman equation. This optimization is almost always performed off-policy, which means that each update can use data collected at any point during training, regardless of how the agent was choosing to explore the environment when the data was obtained. The corresponding policy is obtained via the connection between Q^* and π^*: the actions taken by the Q-learning agent are given by

$$a(s) = \arg\max_a Q_\theta(s, a)$$

Examples of Q-learning methods include

- DQN, a classic which substantially launched the field of deep RL,
- and C51, a variant that learns a distribution over return whose expectation is Q^*.

Trade-offs Between Policy Optimization and Q-Learning. The primary strength of policy optimization methods is that they are principled, in the sense that you directly optimize for the thing you want. This tends to make them stable and reliable. By contrast, Q-learning methods only indirectly optimize for agent performance, by training Q_θ to satisfy a self-consistency equation. There are many failure modes for this kind of learning, so it tends to be less stable. But, Q-learning methods gain the advantage of being substantially more sample efficient when they do work, because they can reuse data more effectively than policy optimization techniques.

Interpolating Between Policy Optimization and Q-Learning. Serendipitously, policy optimization and Q-learning are not incompatible (and under some circumstances, it turns out, equivalent), and there exist a range of algorithms that live in between the two extremes. Algorithms that live on this spectrum are able to carefully trade-off between the strengths and weaknesses of either side. Examples include

- DDPG, an algorithm which concurrently learns a deterministic policy and a Q-function by using each to improve the other,
- and SAC, a variant which uses stochastic policies, entropy regularization, and a few other tricks to stabilize learning and score higher than DDPG on standard benchmarks.

What to Learn Model-Base Unlike model-free RL, there aren't a small number of easy-to-define clusters of methods for model-based RL: there are many orthogonal ways of using models. We'll give a few examples, but the list is far from exhaustive. In each case, the model may either be given or learned.

Background: Pure Planning. The most basic approach never explicitly represents the policy, and instead, uses pure planning techniques like model-predictive control (MPC) to select actions. In MPC, each time the agent observes the environment, it computes a plan which is optimal with respect to the model, where the plan describes all actions to take over some fixed window of time after the present. (Future rewards beyond the horizon may be considered by the planning algorithm through the use of a learned value function.) The agent then executes the first action of the plan, and immediately discards the rest of it. It computes a new plan each time it prepares to interact with the environment, to avoid using an action from a plan with a shorter-than-desired planning horizon.

- The MBMF work explores MPC with learned environment models on some standard benchmark tasks for deep RL.

Expert Iteration. A straightforward follow-on to pure planning involves using and learning an explicit representation of the policy, $\pi_\theta(a|s)$. The agent uses a planning algorithm (like Monte Carlo Tree Search) in the model, generating candidate actions for the plan by sampling from its current policy. The planning algorithm produces an action which is better than what the policy alone would have produced, hence it is an ”expert” relative to the policy. The policy is afterwards updated to produce an action more like the planning algorithm’s output.

The ExIt algorithm uses this approach to train deep neural networks to play Hex. AlphaZero is another example of this approach.

Data Augmentation for Model-Free Methods. Use a model-free RL algorithm to train a policy or Q-function, but either 1) augment real experiences with fictitious ones in updating the agent, or 2) use only fictitous experience for updating the agent.

- MBVE is an example of augmenting real experiences with fictitious ones.
- World Models is an example of using purely fictitious experience to train the agent, which they call ”training in the dream.”

Embedding Planning Loops into Policies. Another approach embeds the planning procedure directly into a policy as a subroutine so that complete plans become side information for the policy while training the output of the policy with any standard model-free algorithm. The key concept is that in this framework, the policy can learn to choose how and when to use the plans. This makes model bias less of a problem, because if the model is bad for planning in some states, the policy can simply learn to ignore it.

3 Related work and State-of-the-art

The state of the art of robotic manipulation is extensive mainly due to two reasons, the large number of approaches and technologies that can be applied, and the vast number of systems required for robotic manipulation, each of which due to its complexity are enough to be considered a central point as an object of research study.

[3] Deals with the main topics that compose robotic manipulation: Object and environment representations, transition models, skill policies, characterizing skills by preconditions and effects, compositional and hierarchical task structures. These topics include different approaches and techniques, although not all of them are applicable to this project due to time limitations for the proposed approach, some of these tools and algorithms are applicable for this project. All these topics are essential for robotic manipulation, and will be treated in this thesis project, some of them more than others, we will focus our attention in skill policies, but also we will need to identify the objects and environment.

3.1 Reinforcement learning algorithms

To learn motor behavior policies is intent to use Reinforcement Learning (RL) algorithms. Some of the most successful and that will be tested for this work, are for example, Deep Deterministic Policy Gradient (DDPG) [33], where they adapt the successful ideas of Deep Q-Learning (DQN) [34], to the domain of continuous actions. Another successful RL algorithm is the Proximal Policy Optimization Algorithm (PPO) [35], which is proposed as a new family of policy gradient method for reinforcement learning, and presents some improvements with respect to the Trust region policy optimization (TRPO) algorithm. [36]. In 2018, a Model-free deep reinforcement learning algorithm is presented, called Soft Actor-Critic (SAC)

[37], based on the maximum entropy reinforcement learning framework, combining off-policy updates with a stable stochastic actor-critic formulation.

3.2 Object recognition for robotic manipulation

We will also need to identify the objects and find an adequate action and environment representation. In order for the robot to be able to perceive and interactive work with its environment. In the state of the art of robotics there are some works that deal with this topic.

Pasquale et al., 2015 [20] *Recognition with Off-the-shelf Deep Conv Nets;* is focused on demonstrating that in the context of a humanoid robot (iCub) it is possible to learn to recognize various objects, using a multi-layer deep convolutional network base provided by Caffe's BVLC Reference CaffeNet library. They use motion-based segmentation to identify the object and it is voice tagged with a human expert.

Bogun et al., 2015 [21] *Object recognition from short videos for robotic perception;* argues that movement can be used in robotics to aid in object recognition. For example, in a robotic arm with a camera mounted, a short video (5 frames) can help recognize the object as it approaches. They developed a method based on Recurrent convolutional networks (RNNs) that use Long Short-Term Memory(LSTM) in their convolutional layers to capture motion information.

Lenz et al., 2015 [22] *Deep learning for detecting robotic grasps;* poses the problem of robotic manipulation from a scene view with RGB-D, the two main challenges posed by the author are the identification of a large number of objects to be taken, so it presents a two-step cascaded system method that uses two deep networks, where the detections of the first network are subsequently evaluated by the second, the first network contains few characteristics, which is faster, and the second evaluates more features but slower than the first one. The second biggest challenge is handling the multimodal inputs, so they present a weight regularization structure based on a multimodal regularization group. The author argues that using RGB-D presents an important advantage when performing grasp activities, as opposed to only using 2D images. They present as one of their contributions a new method to handle multimodal information in the context of feature learning.

3.3 Robotic manipulation as a single centralized agent

Redmon et al., 2015 [23] *Real-time grasp detection using convolutional neural networks;* just as Lenz Redmon addresses the problem of grasping using RGB-D. The difference is that uses a single structure Neural network to identify graspable points, obtaining better results in time and precision. The author uses AlexNet by replacing the blue channel information with the depth information.

Sung et al., 2016 [24] *Robobarista: Learning to manipulate novel objects via deep multimodal embedding;* Derived from the work of Lenz [22], Sung et al. proposes a Robotbarista that integrates several systems, from the identification of objects and the transfer of certain characteristics for manipulation, to a natural language system to give orders to the robot, so that the robot manages to make a latte with appliances that it had never seen before. The authors present an algorithm that uses knowledge from a proof base to infer manipulation trajectory, given point-clouds, and natural language instructions.

Lee et al. 2019 [38] *Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection;* proposes contact-rich manipulation in unstructured environments, using haptic and visual feedback.

Levine et al., 2018 [25] *Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection;* Vision can be used in many ways, in this work a monocular camera is proposed mounted outside the robotic arm at the top so that it can monitor its movements, using two neural networks, and reinforcement learning the robot learns to perform grasping activities. The first network proposes commands to the robotic arm and the second network tries to maximize the grasp probability using a heuristic and a cross-entropy method. This procedure uses a large amount of data provided by several robotic arms, similar to each other, and the learning used by one type of arm can be successfully transferred to others.

3.4 Robotic manipulation as multi-agent approach

The focus of this work is on the multi-agent methodology, so it is relevant to mention some outstanding works. The multi-agent approach has been studied in the literature with some variations, some of the most relevant refer to the way in which the agents are coordinated, being able to be completely independent, or through a master coordinator that can give priority to some agents.

Shahid et al., 2021 [10] *Decentralized multi-agent control of a manipulator in continuous task learning;* Treats the multi-agent problem using a meta-agent that coordinate other agents, in this work they have two agents, one corresponding at the first part of a panda robot and the second one to the second part, they use (PPO and SAC agents) the meta agent that oversee the process, gives more importance to one of the agents. The meta-agent is a high-level policy, and decides which policy should be executed.

Verginis et al., 2017 [11] *Distributed cooperative manipulation under timed temporal specifications;* Treats the problem of cooperative manipulation of a single object, using N robotic agents under a local goal specification, given as metric interval temporal logic. That is a model-free control protocol for the trajectory tracking.

Camacho et al., 2022 [12] *A Reinforcement Learning Decentralized Multi-Agent Control Approach exploiting Cognitive Cooperation on Continuous Environments;* In this work the authors treat the multi-agent control approach using a new reinforcement learning setting. They use a two virtual agents that share the same environment, and control a single avatar but have access to complementary details necessary to finish the task.

Doriya et al., 2015 [13] *A brief survey and analysis of multi-robot communication and coordination* Presents a survey of multi-robot communication and coordination, this survey focus its attention to the problem of robot navigation, and the way multi robots can coordinate to achieve tasks as a team.

Gronauer et al., 2022 [14] *Multi-agent deep reinforcement learning: a survey* Review the advantages of deep reinforcement learning for the multi-agent approach. An analyze of the structure and training schemes is made. Also they consider emergent patters of agent behavior, and show some challenges that arise in the multi-agent domain.

Rizk et al., 2019 [15] *Cooperative heterogeneous multi-robot systems: A survey;* Is a survey about multi-agent systems (MAS) they focus its attention on the challenges of MAS sub-fields including task decomposition, coalition formation, and task allocation.

Feng et al., 2020 [17] *An overview of collaborative robotic manipulation in multi-robot systems;* It's an overview of the state of the art development of collaborative robotic from the perspective of modeling, control and optimization. As the coordination of multiple fixed manipulators, mobile robots and mobile manipulators.

3.5 Analysis of the state of the art

Related to a centralized approach, various authors have dedicated their efforts to strengthen robotic manipulation and optimize training times by implementing the most successful reinforcement learning algorithms, achieving desirable behaviors, but it's training times are quite long. With a decentralized approach through the use of multi-agents, several authors have focused their attention on using several robots to cooperate with each other, assigning an agent to each one. And although they have been successful these robots are independent, this means that there is no link that unites them to part of the ground. Some other authors have tried to implement multiple agents to the same robotic arm, without much success, because they have decided to use an agent for each link.

The proposal of this work differs from the state of the art, in that it is proposed to use a single robot and divide it into different agents, each agent will be trained independently and later they will work together to carry out manipulation activities. The segmentation that will be carried out, differs from other authors in that we segment the robot into strategic parts, so that the agent is capable of using a segment made up of several robotic links, for example the gripper, made up of several fingers. This proposal also differs from other authors in that it proposes the modularization of the segments, so that entire segments can be interchanged and function properly.

4 Research Proposal

The methodology followed to achieve robotic manipulation using a multi-agent approach and a decentralized coordination system is presented.

4.1 Methodology

Some other authors propose a system with a single motor control agent for a specific task, or a multi-agent coordination system that use an agent for every robot in the environment in order to cooperate to achieve the task. This project aims to explore a step by step, and a modular approach, it is intended that motion planning learning takes place in different stages, segmenting the robotic arm into 2 parts and considering each one an independent agent, the parts that make up the arm and the gripper. These systems will independently learn motor control, and will subsequently be coordinated by a third agent.

Parallel to the creation of the multi-agent motor control system, a hierarchical task control system is proposed, the robotic manipulation tasks can be seen as a path that the object to be manipulated has to follow, for example, taking an object from a point A to point B can be seen as the path to be traveled, in this way a task can be accomplished. By creating a hierarchical system, the path planning agent according to the task will coordinate with the motor control system to perform the tasks.

In order to make easier the process of development and implementation of the control algorithms for robotic manipulation, first a 2D simulator will be developed on which the control and coordination algorithms will be designed, later on the simulator provided by Kinova a camera with depth sensors will be adapted, then algorithms developed in the 2D simulator will be tested on it, finally the Kinova Jaco robotic arm will be fitted with an Intel-Realsense camera and the algorithms designed and tested in the Kinova simulator will be adapted.

4.1.1 Motor learning

The first part of the learning process for manipulation activities will be motor learning, which includes the primary skills to control the movements of the robotic arm at a basic level joint by joint, the same applies to the gripper.

The approach to learning motor activities is to separate the robot into two subsystems governed by a different agent each. The first agent will be made up of the longer sections of the robotic arm, and the second will be the gripper only.

Each of these agents will have their own reinforcement learning system based on the most successful algorithm resulting from testing some of the most successful recent algorithms such as Soft Actor-Critic (SAC), Proximal Policy Optimization (PPO), Depp Deterministic Policy Gradient DDPG, to name a few.

2D: Reinforced learning algorithms will be tested in the developed 2D simulator, the learning algorithm will receive an image of the environment with the robot and the target, as well as the angular positions of the joints, with the training it is expected that the agent can carry the robot to the target using spatial position and joint position commands.

3D: Kinova provides a simulator of its robotic arms, the simulator uses ROS and Gazebo, to which the necessary modifications can be made to work in this project, a camera will be adapted to the simulator mounted on the joint between the gripper and the arm robotic.

The idea is that the camera mounted on the arm uses artificial depth vision to generate a cloud of points of the environment, which will deliver the motor control agent together with the spatial and angular position of the components of the robotic arm, in similarity to the algorithm developed in the 2D simulator. The most suitable machine learning algorithm tested in the 2D system will be applied to generate the motor control.

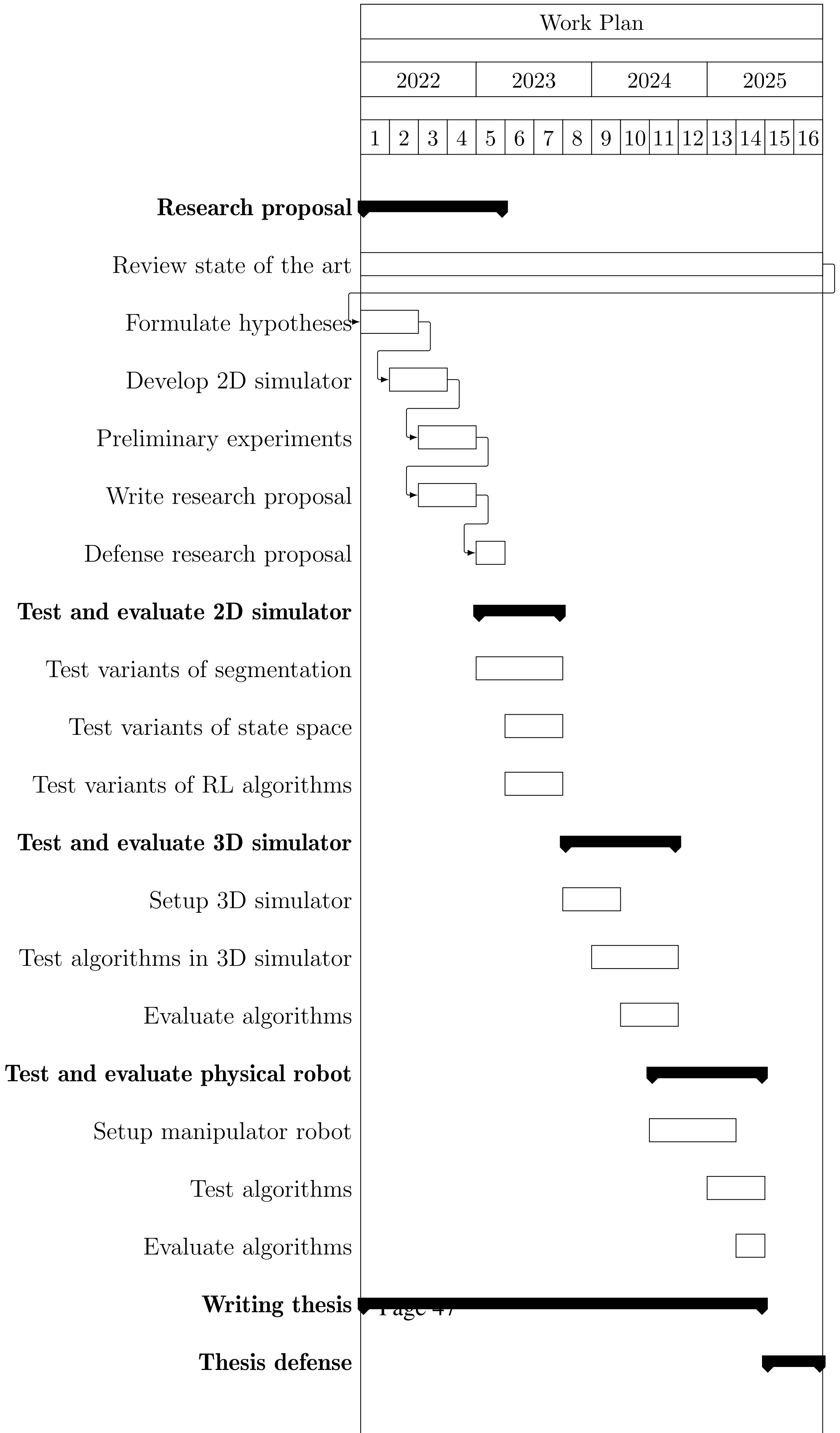
Real robotic system: Finally, the algorithms developed and tested in the simulators will be implemented in a real system, the Kinova-Jaco robotic arm will be used, to which a Real-Sense camera will be mounted in the joint that joins the gripper with the rest of the arm using 3D printed supports.

4.1.2 Coordination system

To coordinate the agents (sections of the robotic arm and base) so that they can carry out joint manipulation activities, a decentralized coordination system is proposed. Since it is not necessary for each agent to receive data from the entire system, the relevant information for each agent is the neighborhood information. All the motor control agents will receive information about their position in space and the position of the previous agent, referring to the previous one that is closer to the robotic base. All the agents will propose new positions for the last joint of the prior agent, in such a way that they will reach a consensus regarding the position that can be achieved, the agents can even send information about positions that they cannot reach, after several iterations the agents are expected to reach a stable position that guarantees the completion of the task.

4.2 Work Plan

The work plan for the next three years will be presented in the next table, the graph is divided by years, and each year in quarters. The focus of the next year is test and evaluate RL algorithms in the multi-agent approach, as well as testing some segmented setup. In the third year the most successful RL algorithms and segmented scheme will be implemented in a 3D environment, using the simulator gazebo-kinova, in this year and beginning of the fourth one the algorithms will be tested in the real robot, a virtualization of the environment and arm will be developed in this time. Throughout the next two and a half years will be used to write articles and the final thesis document. The state of the art will be reviewed constantly during the time of this PHD.

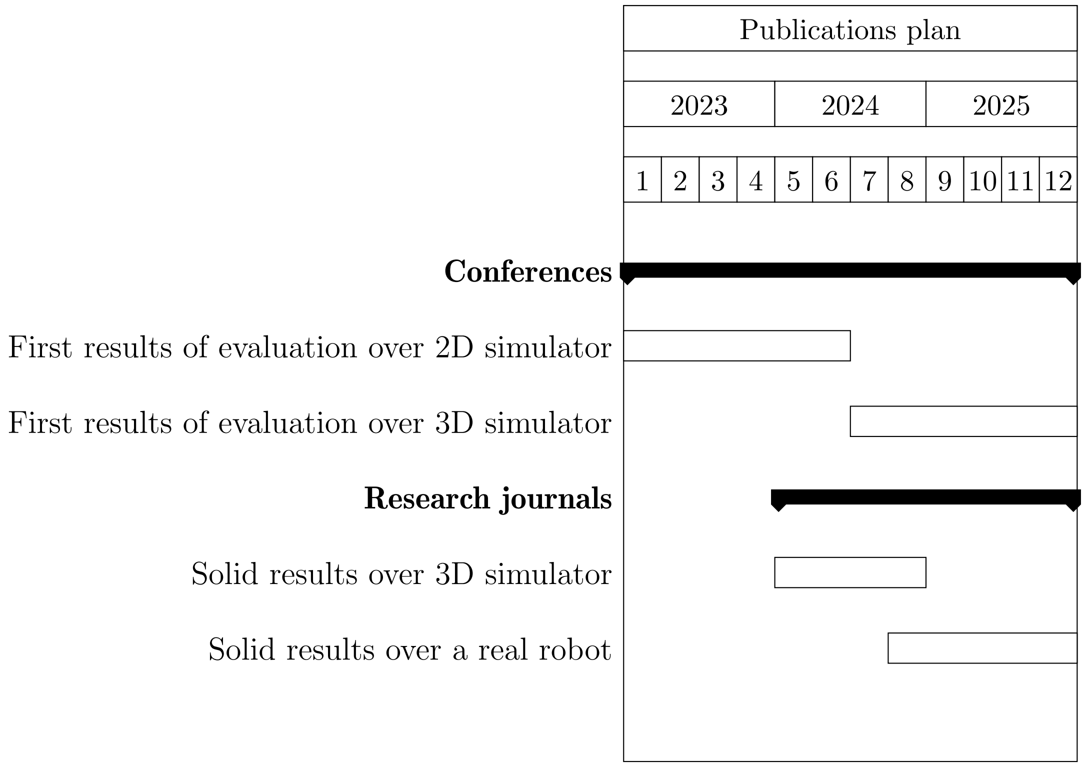

Work Plan
2022
2023
2024
2025
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Research proposal
Review state of the art
Formulate hypotheses
Develop 2D simulator
Preliminary experiments
Write research proposal
Defense research proposal
Test and evaluate 2D simulator
Test variants of segmentation
Test variants of state space
Test variants of RL algorithms
Test and evaluate 3D simulator
Setup 3D simulator
Test algorithms in 3D simulator
Evaluate algorithms
Test and evaluate physical robot
Setup manipulator robot
Test algorithms
Evaluate algorithms
Writing thesis
Thesis defense

4.3 Publications Plan

For the following years it is planned to participate in two congresses presenting preliminary results of the tests carried out in the 2D and 3D simulators respectively. The first participation within the following six quarters and the second in the subsequent six. Some of the objective congresses are ICRA (IEEE International Conference on Robotics and Automation) and IROS (IEEE/RSJ International Conference on intelligent Robots and Systems)

It is also planned to present two articles in journals dedicated to robotics, in the first article the results of the evaluation of the multi-agent system will be presented, using the Kinova 3D simulator, after carrying out exhaustive tests in terms of segmentation, learning and coordination. In the second article, results of the physical robot performing robotic manipulation will be presented, comparing multi-agent learning with centralized learning. It is planned that the first article will be presented in the third year and the second in the fourth year. Some of the objective journals in which the results of this research will be sent are the following. Autonomous Robots, Robotics and Autonomous Systems, IEEE Transactions on Robotics.

This research is not limited to the congresses and journals mentioned above, in case this research could be relevant to any other congress or journal will be considered.


Publications plan
2023
2024
2025
1
2
3
4
5
6
7
8
9
10
11
12
Conferences
First results of evaluation over 2D simulator
First results of evaluation over 3D simulator
Research journals
Solid results over 3D simulator
Solid results over a real robot

5 Preliminary Results

5.1 Presentation of preliminary results

The first part of the research project objectives corresponds to testing machine learning algorithms using the multi-agent approach, in order to test the algorithms a 2D simulator was designed with different environments. Also at this stage, a segmentation configuration of the robot was tested, which is to split it into two parts, the first part corresponds to the robotic base, the second part corresponds to the robotic arm. The SAC algorithm was used to perform reinforcement learning.

These first experiments with a 2D simulator, using the robot separated into two parts, aim to prove that a multi-agent system can learn in less time than a centralized system. This simple simulator will allow to test some of the most successful state-of-the-art RL algorithms, both in the decentralized and centralized systems, and compare training times, in addition to allowing future segmentation tests in different sections of the robot. Also to allow to demonstrate that the sections can be interchanged and in this way to test the coordination system.

In the first test environment there are 3 fundamental elements, the robot, the environment and objective. Three independent training exercises were carried out, in the first exercise, robotic base was trained, with this purpose the environment was adapted, mainly so that the base had a clear objective to reach, in this case an analysis was made to identify in which way the base could achieve its goal, so that the robotic arm agent could also achieve it's own objective.

The rules for this environment were created in the following way, the robotic base cannot touch the edges of the environment, in that case it would be considered a collision, and that is unwanted, within reinforced learning a collision corresponds to a negative reward, and was assigned a value of -50. The second rule of this

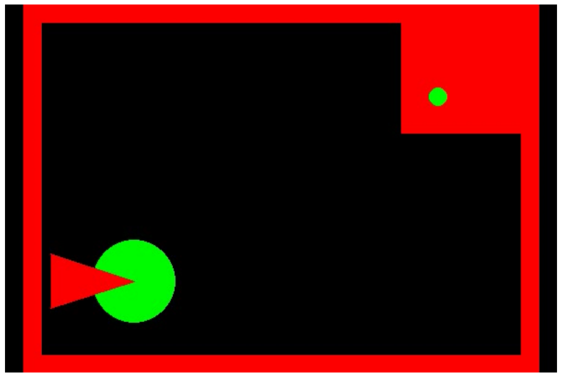


Figure 4: Robotic base in training environment at start point

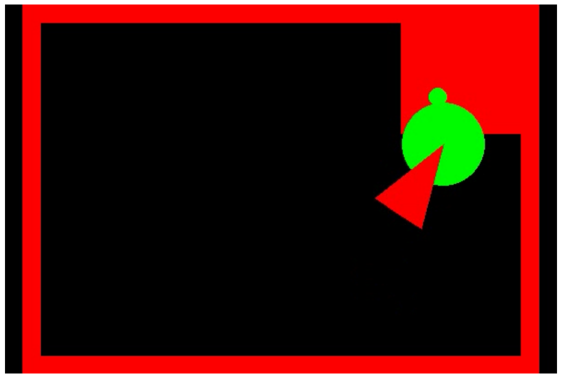


Figure 5: Robotic base in training environment reaching the objective

environment is that the robotic base has to lead the robotic arm to achieve its goal, which is to touch the target point, if the robotic arm can move around a fixed point over the robotic base, then it is clear that a circle can be generated around that point and when any point in that space touches the objective point, in that case the objective of the base can be considered fulfilled.

Figures 4 and 5 show the training environment of the robot segment called robotic base, the first image shows the starting point of the training and the second image shows the robotic base reaching its goal.

The training ended until the robotic base performed 20,000 episodes satisfactorily, this means that the robotic base reached its goal. To achieve this, the training lasted 440,018 frames, one frame corresponding to performing an action and saving

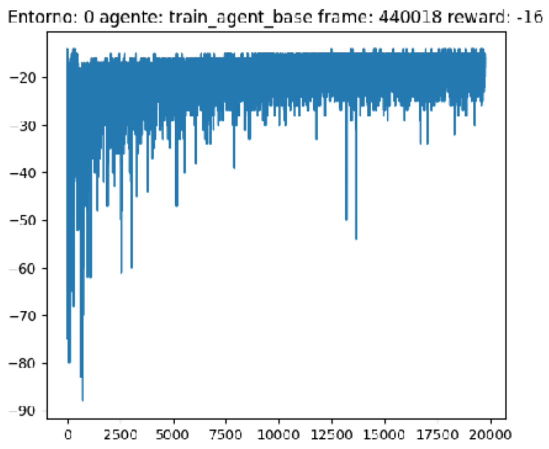


Figure 6: Base segment training episode rewards

the resulting image as data. Every time the robot performs an action it receives a reward of -1, if the action leads it to collide with the environment then the reward corresponds to -50 but if the base reaches its goal then the reward is 50. Figure 6 show the number of steps it take to reach the goal, such as it can be seen, at first it got very negative rewards, this means it takes too many steps to reach the goal, for example at the beginning it takes between 80 and 90 steps to reach the goal, but in the last episodes it only took 16 steps.

The next agent to be trained corresponds to the robotic arm, this training was designed differently than the robotic base agent did. The main difference is that this agent was trained to deal with a more general environment, unlike the robotic base which works in a single environment, the robot arm is trained with different goals. The environment arrangement is as follows, the point of the robot arm which is fixed to the robot base is considered as the center, the link at this point can rotate around from this point but not move, the other end of the link to the fixed point is the beginning of the next link, and the end point of this robotic arm is the one that has to touch the target. To consider the environment solved, as shown in figures 7 and 8.

It took this agent 700,005 steps to reach 20,000 successful episodes, as shown

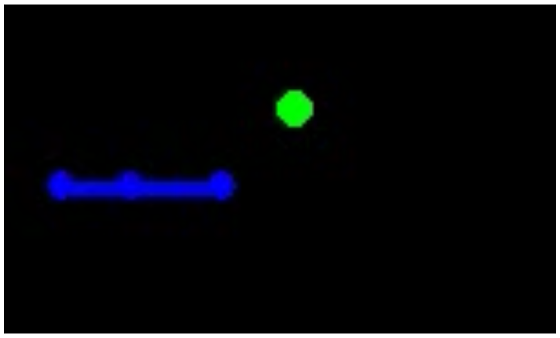


Figure 7: Robotic arm in training environment at start point

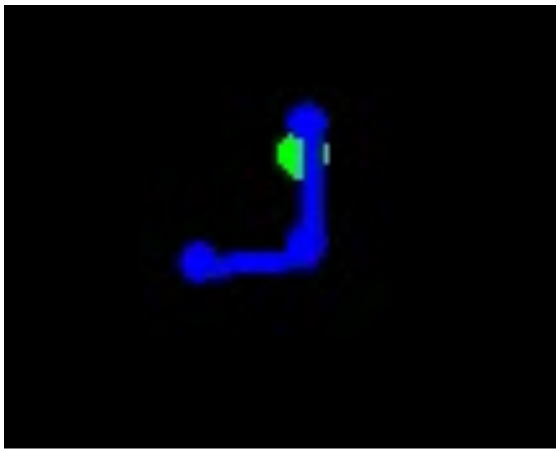


Figure 8: Robotic arm in training environment reaching the objective

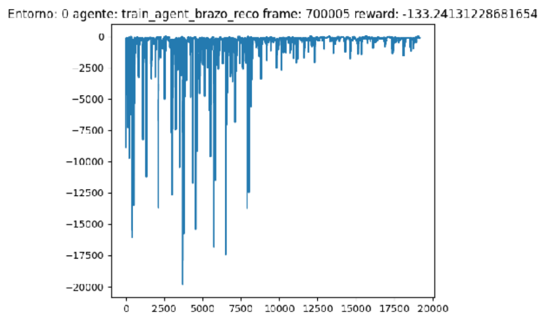


Figure 9: Arm segment training episode rewards

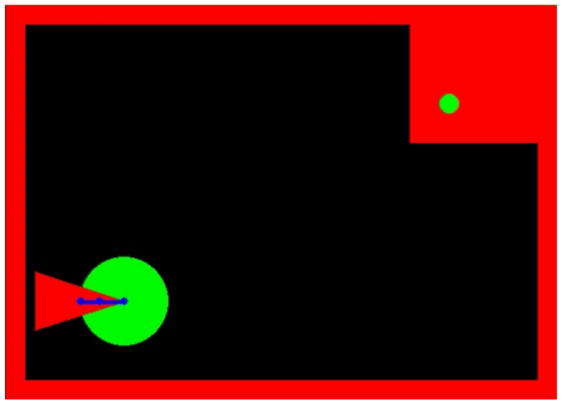


Figure 10: Robot with arm workspace in training environment at start point

in figure 9. It is worth mentioning that it is normal for the arm to take longer to learn its policy than the base because it handles more environments, in case the target changes position unlike the base environment.

The agent training as a whole was also carried out, corresponding to the robotic base and the arm together, in this case the agent has four actions, the first two correspond to the same ones in the robotic base, and the other two correspond to the movement of the robotic arm. We worked with the same environment as shown figures 10, 11 and 12.

Same as the other training sessions, this agent trained until he reached 20,000 successful episodes. But unlike the other trainings, this agent continues in the exploration stage, because the way in which it reaches the objective is not consistent.

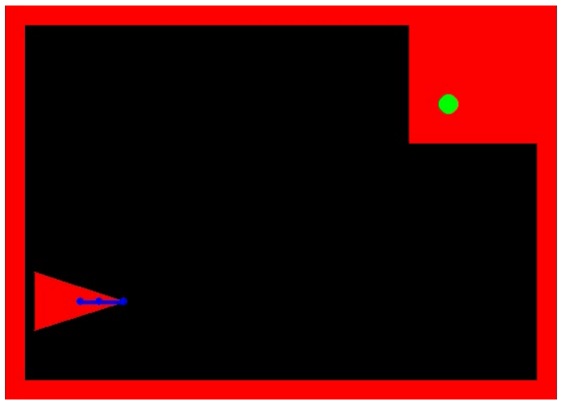


Figure 11: Robot arm in training environment at start point

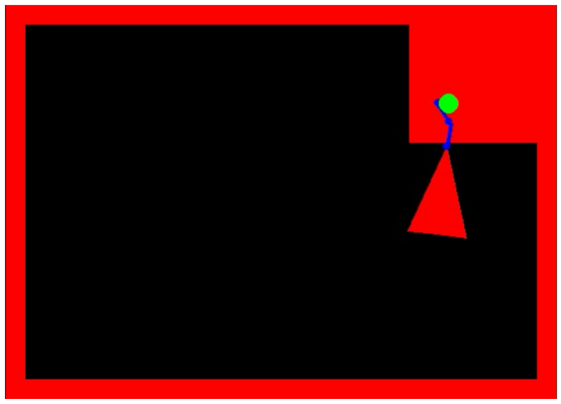


Figure 12: Robot arm in training environment reaching the objective

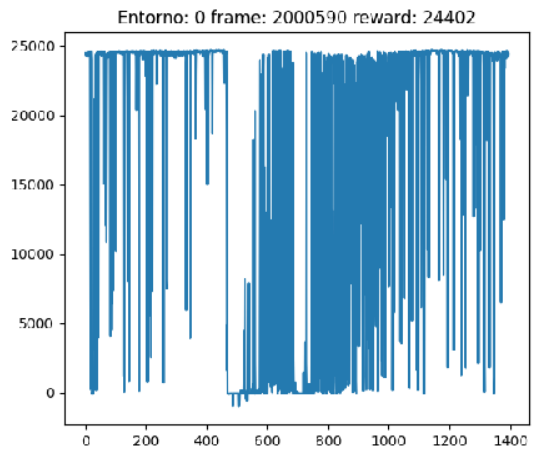


Figure 13: Centralized training episode rewards

This agent required 2,000,590 steps to achieve successful episodes as shown in Figure 13.

In the case of the base and arm trained separately, there is still no complex coordinator, the logic works as follows, when the robotic base reaches its goal, the arm begins to carry out its movements. Both work properly. In the case of the joint training of the base and arm in the same agent, it can be said that it also worked correctly, with the disadvantage that the objective is fixed at one point, so the arm does not have the ability to resolve the environment when the target is in another position. In this case, for the system to resolve the environment, it needs to be trained as a whole, otherwise the arm is trained independently, in which case it is capable of resolving different environments.

5.2 Analysis of preliminary results

These first preliminary results correspond to the first objective of the research work, which seeks to design a simple 2D simulator to test the viability of the approach. From the results obtained, we can deduce that this multi-agent approach is more efficient in terms of training time.

Steps	Agent		
	Segmented Base	Segmented Arm	Centralized system
	440 018	700 005	2000590
Total	1 140 023		2000590

Table 1: Comparison between agent's training steps

Table 1 shows a comparison of the training time of each agent, the decentralized agents took 1,140,023 steps as a whole, while the centralized training took 2000590 steps, both approaches achieved their goal, which is that the final sensor of the arm touches the objective.

The centralized agent took almost twice as long to solve the environment compared to the multi-agent. For later environments, the multi-agent system does not require training all its segments, since the arm agent was trained in a generalized way. So it is expected that for subsequent environments the difference between training times will be even greater.

We can see that the segmented agents differ greatly in their training time, this is because the robotic arm agent was trained to deal with diverse environments, and targets in different positions, this was necessary for the system as a whole to work correctly with the environment. The robotic base was trained in a fixed environment, so the training time was shorter. It should be noted that if the base is placed in another environment it will require retraining, but the robotic arm will not. In contrast, in the centralized approach, the robot has to be completely retrained for each environment.

5.3 Conclusions of the preliminary results

The proposal of a multi-agent system for robotic manipulation purposes is viable.

A first coordination proposal was presented, in which the robotic arm agent was trained with an objective in various positions so that no matter what position the robotic base arrives in, the arm will be able to fulfill its objective.

Although it has been shown that the multi-agent system learns faster than the centralized system, it is believed that the more degrees of freedom the agent has, the difference will be even greater. Because it is believed that the growth of complexity is not linear.

Because each agent can be trained on specialized tasks, the multi-agent approach provides greater flexibility in performing different tasks, transferring skills between tasks, and resolving environments even when not all agents have been trained for it.

6 Final Remarks and Future Work

This work is dedicated to presenting a research proposal related to robotics, specifically robotic manipulation, using a multi-agent approach. By robotic manipulation we understand the ability of a robot to alter the position and orientation of some objects that surround it. The multi-agent approach is focused on segmenting the robot and assigning an agent to each segment, so that each agent can be trained independently using RL and later collaborate to perform manipulation activities.

A brief explanation of what robotic manipulation is and the chosen approach was presented, as well as a review of some novel works in robotic manipulation.

Finally, some preliminary results of the approach followed were presented, where the advantages of using it can be observed, training the systems separately allows greater flexibility when performing tasks in different environments, the separately trained robotic arm allowed the problem to be solved, and even if the position of objective changes, it can solve, otherwise the agent trained as one alone requires new training and the management of a more complex environment. Even so, more tests are required to obtain further validation.

It should be noted that the viability of the chosen approach has been demonstrated, so future work is to continue with this methodology in more complex and challenging environments.

Future work is to perform more tests with the 2D system, testing other RL algorithms, segmentation variations, and the environment. Subsequently, the most successful tests will be implemented in a 3D simulator, the necessary adjustments will be made and later the most appropriate setup will be implemented in a real robot. Where multi-agent learning techniques and coordination with different solid objects of daily use, such as glasses, cups and bottles, will be tested. And with different tasks such as taking, pushing and pulling. The resulting behaviors and the process of learning and coordination will be reported and presented as research articles and as a thesis.

References

[1] B. Zhang, Y. Xie, J. Zhou, K. Wang, and Z. Zhang, "State-of-the-art robotic grippers, grasping and control strategies, as well as their applications in agricultural robots: A review," *Computers and Electronics in Agriculture*, vol. 177, p. 105694, 2020.

[2] M. T. Mason, "Toward robotic manipulation," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, 2018.

[3] O. Kroemer, S. Niekum, and G. D. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *Journal of machine learning research*, vol. 22, no. 30, 2021.

[4] M. Ersen, E. Oztop, and S. Sariel, "Cognition-enabled robot manipulation in human environments: requirements, recent work, and open problems," *IEEE Robotics & Automation Magazine*, vol. 24, no. 3, pp. 108–122, 2017.

[5] A. A. Shahid, D. Piga, F. Braghin, and L. Roveda, "Continuous control actions learning and adaptation for robotic manipulation through reinforcement learning," *Autonomous Robots*, vol. 46, no. 3, pp. 483–498, 2022.

[6] O. Kilinc and G. Montana, "Reinforcement learning for robotic manipulation using simulated locomotion demonstrations," *Machine Learning*, vol. 111, no. 2, pp. 465–486, 2022.

[7] S. Caldera, A. Rassau, and D. Chai, "Review of deep learning methods in robotic grasp detection," *Multimodal Technologies and Interaction*, vol. 2, no. 3, p. 57, 2018.

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

[9] V. Vatsal and N. George, "Augmenting vision-based grasp plans for soft robotic grippers using reinforcement learning," in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pp. 1904–1909, IEEE, 2022.

[10] A. A. Shahid, J. S. V. Sesin, D. Pecioski, F. Braghin, D. Piga, and L. Roveda, "Decentralized multi-agent control of a manipulator in continuous task learning," *Applied Sciences*, vol. 11, no. 21, p. 10227, 2021.

[11] C. K. Verginis and D. V. Dimarogonas, "Distributed cooperative manipulation under timed temporal specifications," in *2017 American Control Conference (ACC)*, pp. 1358–1363, IEEE, 2017.

[12] G. Camacho-Gonzalez, S. D'Avella, C. A. Avizzano, and P. Tripicchio, "A reinforcement learning decentralized multi-agent control approach exploiting cognitive cooperation on continuous environments," in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pp. 1557–1562, IEEE, 2022.

[13] R. Doriya, S. Mishra, and S. Gupta, "A brief survey and analysis of multi-robot communication and coordination," in *International Conference on Computing, Communication & Automation*, pp. 1014–1021, IEEE, 2015.

[14] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, vol. 55, no. 2, pp. 895–943, 2022.

[15] Y. Rizk, M. Awad, and E. W. Tunstel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–31, 2019.

[16] Y. Demazeau, B. An, J. Bajo, and A. Fernández-Caballero, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS

Collection: 16th International Conference, PAAMS 2018, Toledo, Spain, June 20–22, 2018, Proceedings, vol. 10978. Springer, 2018.

[17] Z. Feng, G. Hu, Y. Sun, and J. Soon, "An overview of collaborative robotic manipulation in multi-robot systems," *Annual Reviews in Control*, vol. 49, pp. 113–127, 2020.

[18] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, "Motion planning networks: Bridging the gap between learning-based and classical motion planners," *IEEE Transactions on Robotics*, vol. 37, no. 1, pp. 48–66, 2020.

[19] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[20] G. Pasquale, C. Ciliberto, F. Odone, L. Rosasco, and L. Natale, "Real-world object recognition with off-the-shelf deep conv nets: how many objects can icub learn?," *arXiv preprint arXiv:1504.03154*, 2015.

[21] I. Bogun, A. Angelova, and N. Jaitly, "Object recognition from short videos for robotic perception," *arXiv preprint arXiv:1509.01602*, 2015.

[22] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[23] J. Redmon and A. Angelova, "Real-time grasp detection using convolutional neural networks," in *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 1316–1322, IEEE, 2015.

[24] J. Sung, S. H. Jin, I. Lenz, and A. Saxena, "Robobarista: Learning to manipulate novel objects via deep multimodal embedding," *arXiv preprint arXiv:1601.02705*, 2016.

[25] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International journal of robotics research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[26] O. Alagoz, H. Hsu, A. J. Schaefer, and M. S. Roberts, "Markov decision processes: a tool for sequential decision making under uncertainty," *Medical Decision Making*, vol. 30, no. 4, pp. 474–483, 2010.

[27] F. Garcia and E. Rachelson, "Markov decision processes," *Markov Decision Processes in Artificial Intelligence*, pp. 1–38, 2013.

[28] H. Wang, C. Ma, and L. Zhou, "A brief review of machine learning and its application," in *2009 international conference on information engineering and computer science*, pp. 1–4, IEEE, 2009.

[29] S.-C. Wang, "Artificial neural network," in *Interdisciplinary computing in java programming*, pp. 81–100, Springer, 2003.

[30] J. Koushik, "Understanding convolutional neural networks," *arXiv preprint arXiv:1605.09081*, 2016.

[31] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

[32] G. Thomas, "Markov decision processes," 2007.

[33] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[36] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

[37] 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*, pp. 1861–1870, PMLR, 2018.

[38] M. A. Lee, Y. Zhu, K. Srinivasan, P. Shah, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg, "Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8943–8950, IEEE, 2019.