ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Reinforcement Learning for Robotic Actors in Multi-Agent Setups

by

Yoann Ponti

Master Thesis

Supervisors

Mischa Schmidt Ph.D.
NEC Laboratories Europe GmbH

Prof. Boi Faltings
Artificial Intelligence Laboratory (EPFL)

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

October 2018

Abstract

This exploratory work looks at a decentralised multi-agent reinforcement learning problem in which each agent is trying to maximise a global utility. Agents having only access to partial observations of the environment, a key challenge is coordination. To mitigate the effect of the partial observability, the agents must learn to communicate and share the information required to solve the task. The main objective of this work is to provide an in-depth look at the quality of the learning, the behaviour of the agents and the learned communication protocols.

Acknowledgements

First and foremost, I would like to thank my supervisor Mischa Schmidt Ph.D. as well as Sébastien Nicolas and Anett Schuelke Ph.D. for providing guidance and encouragement in carrying out this project at NEC Laboratories Europe GmbH. I respect and thank Prof. Boi Faltings for agreeing to supervise me and give me the opportunity to do a project in the industry. My thanks and appreciations also go to my colleagues in developing the project and people who have willingly helped me out with their abilities. I would like to particularly thank, Julia G., Cristof K., Shahd S. and Mauricio F. A. for their thorough proofreading and feed backs.

Contents

Abstract **i**

Acknowledgements **ii**

List of Figures **v**

List of Tables **vi**

Abbreviations **vii**

Symbols **viii**

1 Introduction **1**
- 1.1 Contributions . . . 2
- 1.2 Limits . . . 2
- 1.3 Outline . . . 3

2 Background **4**
- 2.1 Markov Decision Processes . . . 5
  - 2.1.1 Learning with complete knowledge of the environment . . . 7
  - 2.1.2 Learning with incomplete knowledge of the environment . . . 9
- 2.2 From single-agent to multi-agent learning . . . 13
  - 2.2.1 Partial Observable MDP . . . 14
  - 2.2.2 Decentralised Learning in a *POMDP* . . . 15
- 2.3 Related Work . . . 16

3 Experiments **18**
- 3.1 Experimental Setup . . . 18
- 3.2 Communication schemes . . . 20
- 3.3 Reward function . . . 21
- 3.4 Training . . . 22
  - 3.4.1 Action selection and exploration . . . 22

4 Results & Discussion **24**
- 4.1 Learning approach comparison . . . 24
- 4.2 Risk aversion . . . 26

4.3 Communication Analysis . . . 27
4.3.1 Influence of communication reward . . . 27
4.3.2 Vocabulary size . . . 31
4.4 Unreliable channel . . . 34
4.5 Fully agent controlled . . . 36

5 Future Work and Conclusion **37**
5.1 Future research . . . 37
5.2 Conclusion . . . 37

A Determination of the grounding of the protocol **38**

B Architecture overview **41**
B.1 World . . . 42
B.1.1 Buffer . . . 42
B.1.2 Conveyor . . . 42
B.1.3 Robotic Arm . . . 43
B.2 Scenarios & Runners . . . 44
B.2.1 Learning Cycle . . . 45

C Robotic arm Python wrapper **46**
C.1 Robotic interface . . . 46
C.1.1 Robotic arm . . . 47
C.2 Architecture design . . . 47
C.2.1 Overview . . . 47

Bibliography **49**

List of Figures

2.1 The agent-environment interactions in a Markov decision process 5

3.1 Scenario overview 19
3.2 Comparison of process flow between instantaneous and delayed communication [1] 20

4.1 Behaviour of agents relative to training method 25
4.2 Average risk aversion relative to the time-reward 26
4.3 Average risk aversion relative to the discount factor 26
4.4 Influence of r_{com} on optimality of learning 27
4.5 Influence of r_{com} on the behaviour of the agents 27
4.6 Ideal protocol for $|V| = 2$ 30
4.7 Misinterpreted protocol for $|V| = 2$ 30
4.8 Grounding of communication relative to r_{com} for $|v|$ =2 31
4.9 Empirical distribution of protocol types (default parameters) 33
4.10 Non-ideal communication protocol for $|V| = 4$ 34
4.11 Ideal communication protocol for $|V| = 4$ 34
4.12 Empirical distribution of protocol types relative to the discount rate . . . 35
4.13 Empirical distribution of protocol types (best parameters) 35
4.14 Reaction to received communication relative to the drop rate 36
4.15 Optimality relative to the drop rate 36

A.1 Protocol graph related to the intent of the symbols 39
A.2 Final protocol graph related to the example. 40

B.1 Overview of the experiment architecture 41
B.2 Overview of the architecture used to control the robotic arms 44

C.1 Overview of the ROS wrapper architecture 47
C.2 Overview of a controller work-flow 48

List of Tables

4.1 Optimality ratio relative to the training method 24
4.2 Abstracted sub states for $agent_{arm3c}$ 28
4.3 Number of valid vocabulary assignments 32
4.4 Optimality ratios with default hyperparameters relative to vocabulary size 32
4.5 Distribution of protocol type 33
4.6 Valid vocabulary assignments with unreliable channel 35
4.7 Optimality and behaviour with fully decentralised learning 36

A.1 Interpretation of symbols by the feeding agents 39
A.2 Protocol graph related to the interpretation of symbols by the feeding agents 39
A.3 Vocabulary assignment related to the interpretation. 39
A.4 Vocabulary assignment related to the intent. 39
A.5 Vocabulary assignment related to the grounding. 40

Abbreviations

AI	**A**rtificial **I**ntelligence
Dec-POMDP	**D**ecentralised **P**artially **O**bservable **M**arkov **D**ecision **P**rocess
MARL	**M**ulti-**A**gent **R**einforcement **L**earning
MAS	**M**ulti-**A**gent **S**ystem
MC	**M**onte-**C**arlo
MDP	**M**arkov **D**ecision **P**rocess
POMDP	**P**artially **O**bservable **M**arkov **D**ecision **P**rocess
RL	**R**einforcement **L**earning
ROS	**R**obot **O**perating **S**ystem
TD	**T**emporal **D**ifference

Symbols

α, β	step-size parameters
γ	discount-rate factor
s	a state
a	an action
r	a reward
o	an observation (related to a state s)
v	communication symbol
A	set of available actions
S	set of states
O	set of observations
V	set of communication symbols (vocabulary)
t	discrete time step
T	final time step of an episode
G_t	return following time t
π	policy
π^*	optimal policy
$\pi(s)$	action taken in state s under deterministic policy π
$\pi(a\|s)$	probability of taking action a in state s under stochastic policy π
$v_\pi(s)$	value of state s under policy π (expected return)
$v_{\pi^*}(s)$	value of state s under the optimal policy
$q_\pi(s,a)$	value of taking action a in state s under policy π
$q_{\pi^*}(s,a)$	value of taking action a in state s under optimal policy
V	array estimates of state-value function v_π
Q	array estimates of action-value function q_π

Chapter 1

Introduction

Reinforcement learning has seen a lot of improvement since its first introduction and it has now been used to help solve many tasks thought before as unsolvable. For example, reinforcement learning has been used to beat human players at Atari games [1] or master the game of GO [2]. Moreover, reinforcement learning by solving the task of sequential decision making in an unknown environment is particularly well suited to solve tasks where it is difficult to obtain training data (robotics, telecommunication, etc.).

Despite the usefulness of reinforcement learning in many fields, the method can lead to several problems. In an ideal world, any machine learning method should be able to cope with noisy input. Moreover, it should be general enough and scalable. In order to be able to learn, an agent requires feedback about the quality of the action taken. Unfortunately in most of the real-world scenarios, the feedback might be temporally delayed and as a result only impact the preceding states and actions weakly. For example, a robot might have to take several actions before relevant events happen. Several reinforcement learning algorithms assume the environment to be Markovian and that it is irrelevant how it has reached a certain state. Reinforcement learning agents might not know exactly in what state they are or will be after performing an action. Consequently, the preceding assumption is broken. Finally, reinforcement learning can only cope with dynamics that change slowly. This is a fundamental problem. Convergence of the learning might not be achievable if the world changes too fast [3]. Setting up a reinforcement learning problem might reveal itself to be difficult. Whether it is a real-world problem or a simulated one, appropriate state and action spaces need to be determined. In order to cover all possible situations, the state space as well as the available actions might be large. Exploring all possible pairs can then lead to a combinatorial explosion [4, 5]. The situation worsens when multiple agents are involved. The problem becomes exponential in the number of agents [6].

In order to cope with the curse of dimensionality, several solutions have been proposed such as generalising value functions. This is either done using function approximation methods or by interpolation of the searchable value-space. When the "action dimensions are able to operate in parallel and their individual information and resources can be managed separately", the task can be decentralised [6]. It has been shown that decentralised systems are able to achieve similar learning times and comparable or slightly lower performance. To compensate the lack of direct coordination between the agents which can lead to lower performance, inter-agent communication can be used. Communication can help increase the team performance by mitigating two sources of uncertainty. It can help agents by increasing their knowledge about the state of the system as well as the intention of the other agents [7]. The following explores this specific approach in a robotic transport system. In particular, the usage of the learned communication protocols is examined in-depth.

1.1 Contributions

This Master project explores the application of distributed reinforcement learning in a coordination problem involving robotic arms. It explores inter-agent communication in a *Dec-POMDP* environment to enable learning in a fully decentralised fashion. The communication protocols are also examined in order to to gain insight in the influence of the hyperparameters on the solutions.

1.2 Limits

In order to study the behaviour of the agents, several assumptions had to be made. Even though the underlying problem is linked to the physical world, only simulated experiments are used. The problem is sufficiently simple to obtain good understanding of the underlying process and know which state information is relevant. This might not always be possible. This work relies on tabular Q-Learning. It enables faster learning and a clear view of the learning process. Unfortunately, this learning algorithm does not scale to more complex problems. Also the simulated environment makes assumption about the available interaction between the entities. The simulated environment look pass many problems that could arise in a real-world setting.

1.3 Outline

The work is organised as follows. In chapter 2, the theoretical background linked to reinforcement learning and the Markov decision process framework are introduced. Furthermore, the necessary extensions needed by the transition from single agent to a multi-agent system as well as the related works are presented. The chapter 3 presents the experiment setting used in chapter 4. The latter presents, examines and discusses the results. In particular this chapter describes the influence of hyperparameters on the quality of the learned communication. Finally, chapter 5 discusses future work and concludes.

Chapter 2

Background

The goal of machine learning is to avoid solving a specific task by explicitly programming an algorithm. In order to do this, machine learning tries to optimise a performance criterion using example data or past experience. When trying to solve a new task, it might be difficult to design a complete specification of the algorithm behaviour while it might be possible to obtain samples of the expected behaviour or to get feedback on the quality of the behaviour. Machine learning models can be predictive, i.e. make predictions based on new data, or descriptive, when the task of the model is to gain knowledge from data, or both [8].

Reinforcement learning can be loosely related to how we as humans, learn to interact with our environment. For most of our life we do not have access to an explicit teacher, but we do have access to a direct sensorimotor connection to our environment providing feedback. Leveraging this feedback offers an indication about the consequences of an interaction with the environment. This indication can help us determine what to do to achieve a certain outcome. Reinforcement learning is the machine learning approach for learning goal oriented tasks from interaction.

Other approaches to machine learning include *supervised learning* and *unsupervised learning*. In supervised learning, a set of labelled examples provided by an external entity is used for the learning. Although supervised learning has been shown to be highly efficient for certain problems, obtaining the input-output pairs in an interactive problem can be extremely difficult. Indeed, the pairs should be representative of the expected behaviour as well as of all the states in which the agent should act. On the other hand, unsupervised learning finds hidden relationships between unlabelled inputs. Despite reinforcement learning having no direct access to the input-output examples, it should be considered a paradigm in its own right and not a kind of unsupervised

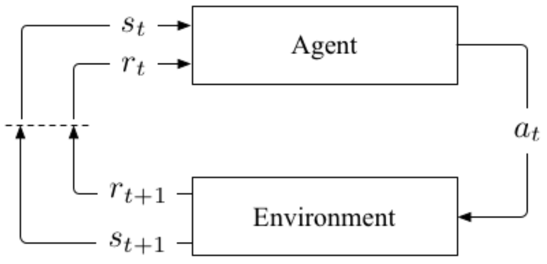


FIGURE 2.1: The agent-environment interactions in a Markov decision process [1].

learning. Indeed, it tries to maximise a reward signal instead of finding hidden pattern in the inputs.

Instead of relying on an oracle to instruct a learner the best action to take, reinforcement learning has to rely on a signal related to the quality of its decision [9, p. 25]. The signal, which indicates what to do, but not how to do it, is expressed by a sanction called a *reward* [10, p. 18]. The reward can be positive, negative or neutral. It might be influenced by the taken action as well as the agent's past actions (delayed impact). The goal of reinforcement learning is thus to optimise a reward function by interacting with its environment (i.e. taking actions) and observing the resulting reward [11]. Actions which have a positive outcome or avoid negative one will be preferred (reinforced). This is why reinforcement learning can be seen as a trial and error search.

2.1 Markov Decision Processes

In reinforcement learning, it is important to make the distinction between an agent and the environment. An agent is a computational mechanism able to learn and to take actions based on external information [12]. The environment is everything else. More precisely, it is everything which cannot directly be modified by the agent [9, p. 50].

The interaction between an agent and the environment is done at discrete time-steps. At each time-step, the agent observes the environment and takes an action accordingly. The environment will then transition to another state and provide a numerical signal, the reward, related to the transition. When a single agent is present in the environment, Markov decision processes (MDPs) can be leveraged to express mathematically the reinforcement learning problem [13]. The interaction of the agent with the environment as depicted in fig. 2.1 creates a sequence of tuples (s_t, a_t, r_t, s_{t+1}) describing one *trajectory* or *episode*. Besides a finite set of states and actions, S and A, the MDP requires a definition of its dynamics. The dynamics of any MDP can be defined by the probability to move to state $s' \in S$ and receive reward $r \in \mathbb{R}$ by taking action $a \in A$ in state $s \in S$.

[1]Figure taken from [9].

$$p(s', r|s, a) = P(s_t = s', r_t = r \mid s_{t-1} = s, a_{t-1} = a)$$

Based on the recorded trajectory, it is possible to calculate the return or the sum of reward from time t to the end of the episode.

$$G_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} = r_{t+1} + \gamma G_{t+1} \text{ with } G_T = 0$$

Because an episode can be infinite (continuing task) or finite (episodic task), a discounting factor γ is introduced in the formulation to avoid unbounded sums. The discount factor also determines the importance of future reward compared to immediate reward.

When interacting with the environment, an agent is expected to choose an action a_t in state s_t according to a certain policy $a_t = \pi(s_t)$. If the agent follows a fixed policy, it is possible to express the expected return using the *state-value function*. The function expresses the amount of reward an agent can expect to receive when starting in state s and following π

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

.

Similarly, the expected return by taking action in state s and then following π is expressed by the *action-value function*,

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

.

Both functions can be used to organise and structure the search for good policies [9, p. 73] and can be calculated iteratively using the Bellman equation [14]. The Bellman equation expresses the relationship between the value of a state and the value of its successor state. Moreover when the MDP is finite, the Bellman equation for the optimal state-value function v^* has a unique solution independent of the policy π. When v^* is available, it is straight forward to define an optimal policy π^* by being greedy with respect to the state-value function.

$$v^{\pi^*}(s) = \max_\pi v^\pi(s) \text{ for } \forall s \in S \tag{2.1}$$

When a complete description of the underlying MDP is available, an optimal policy can be determined without experiencing the environment linked to it. In such cases, *model-based* methods can be used. An example is dynamic programming. More often though, the dynamics of the environment might not be known and must be learned by interacting with the MDP. Such techniques are called *model-free*.

Using the reward as a description of its goal, the objective of the agent is to find an optimal policy π^* which maximises the long term reward that it might be able to accumulate over time [10].

2.1.1 Learning with complete knowledge of the environment

In model-based learning, two main approaches can be used to find an optimal policy. The policy can be directly manipulated with the *policy iteration* method. An other approach is *value iteration*, which looks for the optimal value function [10].

The idea behind policy iteration and value iteration is to ask what happens if, after computing $v_\pi(s)$, instead of selecting $a = \pi(s)$ in s, another action is selected. This is embodied by the action-value function $q(s, a)$. Generalising this idea to all states, a new policy can be computed using

$$\pi'(s) = \arg\max_a q_\pi(s, a) \tag{2.2}$$

.

Using the *policy improvement theorem* [9, p. 78], the new policy $\pi'(s)$ must be better or equal to the initial one, i.e.

$$q_\pi(s, \pi'(s)) = v_{\pi'}(s) \geq v_\pi(s) = q_\pi(s, \pi(s))$$

.

Following now π' instead of the initial policy should give rise to higher or equal return for all states s. Making a new policy that improves on the original one by making it greedy with respect to the value function of the original as in equation 2.2 is called *policy improvement*. The combination of the determination of the value-function with respect to π (*policy evaluation*) with policy improvement, is called policy iteration method (alg. 1).

Algorithm 1 Policy Iteration [9]

```
*Initialisation*:
$V(s) \in \mathbb{R}$ and $\pi(s) \in A$ arbitrarily for all $s \in S$

*Policy Evaluation*:
repeat
    $\Delta \leftarrow 0$
    for all $s \in S$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, \mid v - V(s) \mid)$
until $\Delta < \Theta$

*Policy Improvement*:
$policy\text{-}stable \leftarrow true$
for all $s \in S$ do
    $old\text{-}action \leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r' \mid s, a)[r + \gamma V(s')]$
    if $old\text{-}action \neq \pi(s)$ then $policy\text{-}stable \leftarrow false$
if $policy\text{-}stable$ then
    return $V \approx v^*, \pi \approx \pi^*$
else
    goto *Policy Evaluation.*
```

By contrast, value iteration (alg. 2) combines in a single update one sweep of policy evaluation and one sweep of policy improvement. It does so by avoiding the tracking of the policy and concentrating on finding the optimal value function. When the optimal value function is found, a deterministic policy can be found using the greedy approach as in eq. 2.1.

Algorithm 2 Value Iteration [9]

```
$\Theta > 0$
Initialise $V(s)$, for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $V(terminal) = 0$

**repeat**
    $\Delta \leftarrow 0$
    **for all** $s \in S$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, \mid v - V(s) \mid)$
**until** $\Delta < \Theta$
Output a deterministic policy, $\pi \approx \pi^*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$
```

Both methods can be seen as a version of *Generalised Policy Iteration.* In Generalised Policy Iteration two processes interact. One which makes the value function consistent with the current policy, and the other which makes the policy greedy with respect to the current value function [9, p. 86].

2.1.2 Learning with incomplete knowledge of the environment

When value functions need to be estimated using experience only, two main *model-free* approaches exist. The first one relies on Monte-Carlo experiments using sampled episodes, while the second one can learn directly from raw experience. Both approaches can be used to find optimal policies without any information beside experience [9, p. 98].

The first approach is encompassed by Monte-Carlo Methods. They first generate experience (i.e episodes) by following a policy π and recording the resulting sequence of tuples (s_t, a_t, r_t). The main idea behind the Monte-Carlo approach is to estimate the state-value function based on the average return G_t observed after visiting that state.

$$V(s_t) \longleftarrow V(s_t) + \alpha[G_t - V(S_t)]$$

This means that instead of computing the value of each state by using the model of the MDP, it relies on the empirical average return starting in that specific state [9, p. 115]. Monte-Carlo methods are not bootstrapped which means they do not build the estimate for each state based on other estimate. With those methods, the full sequence of received rewards is known in advance. Therefore, it is possible to know the long term return from a specific state. Without access to the model of the MDP, the value for each action in a specific state must be estimated. Unlike for model-based methods, the single step look-ahead used to determine the best action cannot be made [9, p. 96].

When experiencing the environment, choosing which action to take in a certain state can have a big impact on the learning process. Indeed, an agent can either exploit his knowledge and take actions that will yield high reward in the short term or take a sub-optimal route now, with the hope of achieving better results in the future. This is called the *exploration and exploitation trade-off*. Finding the right balance between the two has been shown to be problematic [9, 15]. To be certain of the convergence of the Monte-Carlo methods, all actions in each state must be selected with a non zero probability to ensure constant exploration of the state space and all state transition [9, p. 97].

There are two ways to find an optimal policy and ensure the exploration. Either by *on-policy learning*, where the policy used to make the decisions is evaluated and also improved, or by *off-policy learning*, where a different policy than the one used to generate the data is evaluated and improved.

With on-policy learning, the agent commits to always exploring by making the followed policy soft, i.e. $\pi(a|s) > 0 \, \forall s \in S, \forall a \in A$, but gradually shifting closer to a deterministic policy [9, p. 100]. An example of such a policy is the ϵ-*greedy* one, where the greedy action is selected with a probability $1 - \epsilon$, while the rest of the time a random action is selected. By contrast, off-policy learning uses two different policies. One policy that is learned about and that becomes the optimal one and one for exploring.

In place of waiting the end of an episode to know G_t, it is possible to update an estimate based on another learned estimate, i.e. bootstrap. This is call *temporal difference* (TD) learning. Instead of using the Monte-Carlo methods update, TD methods change the update rule to

$$V(s_t) \longleftarrow V(s_t) + \alpha[R_t + \gamma(V(s_t + 1) - V(s_t)]$$

.

Both update methods follow the canonical form

$$\text{new-estimate} \longleftarrow \text{old-estimate} + \text{step-size} \times [\text{target} - \text{old-estimate}]$$

where it is assumed that the target indicates a desirable direction in which to move [9, p. 30].

There is no need to wait until the end of an episode to know the update of $V(s_t)$ by using the second update rule. Only the next time step is required by TD methods. The $[R_t + \gamma(V(s_t + 1) - V(s_t)]$ part of the update can be seen as an error (TD *error*) which measures the difference between the estimated value of s_t and the better estimate $R_t + \gamma(V(s_t + 1)$ [9, p. 121]. It can be implemented online in a fully incremental fashion [9, p. 124]. If the update is made after every transition, the method is called *TD(0)*. It has been proven that by following a fixed policy π, $TD(0)$ will converge to v_π, if the learning gain α decreases sufficiently slowly towards zero [16]. A comparison between Monte-Carlo and TD methods can be found in [9, p. 123].

By following the pattern of generalised policy iterations, it is possible to determine an optimal policy. Just like for Monte-Carlo methods, the trade-off between exploration and exploitation can be approached from two different angles.

The action-value function $q_\pi(s, a)$, rather than the state-value function, is estimated using

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

which can be executed after every transition. This update method is the base for the on-policy *Sarsa* control algorithm. Sarsa converges to an optimal policy and an optimal action-value function as long as the policy converges at the limit to a greedy policy. This can be achieved by using an ϵ-greedy policy where $\epsilon = 1/t$ [9, p. 129]. The Sarsa algorithm is an on-policy method because the action selection is done relative to the same policy as the one that is improved. The pseudo-code for the algorithm can be found in alg. 3.

Algorithm 3 Sarsa (on-policy TD control) [9]

```
$\alpha \in (0,1], \epsilon > 0$
Initialise $Q(s,a)$, for all $s \in S$, $a \in A(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

for all episodes do
    Initialise $S$
    Choose $a_t$ using policy derived from Q (e.g., $\epsilon$-greedy)
    for all steps in episode do
        Take action $a_t$, observe $_t$ and $s_{t+1}$
        Choose $a_{t+1}$ using policy derived from Q (e.g., $\epsilon$-greedy)
        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$,
        $s_t \leftarrow s_{t+1})$, $a_t \leftarrow a_{t+1}$
        if $s_t$ is terminal then
            return
```

Watkins and Dayan presented another approach called *Q-Learning* [17]. Q-Learning directly approximates the optimal action-value function independently of the policy being followed. For convergence to be guaranteed, the followed policy must enable all state-action pairs to be updated.

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha[r_t + \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Q-learning is an off-policy control method because the action selection is taken with respect to an ϵ-greedy algorithm while the action value function update is determined using a different policy i.e. one that is directly greedy with respect to the action value function (indicated by $\max_a Q(s_{t+1}, a_{t+1})$).

Algorithm 4 Q-learning (off-policy TD control) [9]

```
$\alpha \in (0, 1], \epsilon > 0$
Initialise $Q(s, a)$, for all $s \in S$, $a \in A(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

for all episodes do
    Initialise $S$
    Choose $a_t$ using policy derived from Q (e.g., $\epsilon$-greedy)
    for all steps in episode do
        Take action $a_t$, observe $r_t$ and $s_{t+1}$
        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$,
        $s_t \leftarrow s_{t+1})$
        if $s_t$ is terminal then
            return
```

Up to now all methods presented have been relying on estimating a value-function in order to find an optimal policy. Another approach is to learn directly a parameterized policy $\pi(a|s, \theta)$ which select actions without consulting any value function [9, p. 323]. This kind of techniques are called *policy gradient methods*.

Learning the parameters of the policy is done via the gradient of some performance measure $J(\theta)$. To ensure convergence, it is required that the policy never becomes fully deterministic. This can be achieved by using parameterized numerical preferences for each state-action pair. An example of such parameterization is the softmax distribution.

Methods which learn both an estimation of the policy and the value function are called *actor-critic methods*.

2.2 From single-agent to multi-agent learning

Until now, only single-agent environments were considered. When a group of autonomous, interacting entities share a common environment, the system is transformed into a multi-agent system (MAS) [15]. Also, the tuple resulting from a single step in an episode has now the form $(s, a^1, ..., a^M, r^1, ..., r^M)$ with $M = \#agents$ [6]. If all agents receive the same reward, they have the same goal and the task is considered fully cooperative. If it is not the case, the task can be competitive or a balance between the two [12].

Learning in a MAS through interaction is called multi-agent reinforcement learning (MARL). Even though a MARL problem can be solved using a single learner, it provides an alternative perspective on systems that are originally regarded as centralised [5]. This might in turn help solve real-world problems too complex or big to solve centrally.

A generalisation of the Markov decision process for the multi-agent case can be found in game theory. In the framework of *stochastic games*, players or agents are thought of as rational individual which try to maximise their own payoff. A stochastic game can either be a *static (stateless) game* or a *stage game*. A static game is related to the bandit problem where no state information is available. When a static game is played repeatedly, it becomes a repeated game. The repetition enables the agents to learn about other agents as well as the reward function. When state information is available, the stochastic game is called a stage game [5]. If the assumption of agent rationality is respected, the *Nash equilibrium* can be used to study the behaviour of agents [15]. A Nash equilibrium is achieved when the individual strategy is the best response to others strategy. It corresponds to a status quo, no agent has anything to gain by changing only their own policy unilaterally [5]. In certain case, reinforcement learning problems can be seen as a referential game or signalling game [18]. In this framework, two players often called *sender* and *receiver* interacts. The sender looks at the actual state and choose a message from a set V. The receiver then observes the signal and choose an action. If the actual state is correctly interpreted, both agents receive a positive reward.

2.2.1 Partial Observable MDP

Until now, it was assumed that all the necessary information that would influence an agent's decision process was available. As stated by Altman & Eitan, in a Markov decision process, the probability of each state and reward depends only on the immediately preceding state and action. Thus, the state must have all the information that makes a difference for the future [19].

In certain cases, an agent might have to interact and make decisions based on incomplete information about the state of the environment. This setting has been formalised using the partially observable Markov decision process ($POMDP$) framework [20]. In a $POMDP$, an agent, after taking action a_t, will receive an observation $o_{t+1} \in O$ related to the state s_{t+1} by the probability distribution $p(o_{t+1}|s_{t+1}, a_t)$. In this situation, the observation will generally not uniquely identify the state s_{t+1}. As a result, the learner might be uncertain about the true state of the environment [7].

2.2.2 Decentralised Learning in a *POMDP*

When a group of cooperating agents interacts in a stochastic environment, the decentralised *POMDP* (*Dec-POMDP*) framework can be used. It is a generalisation of the *POMDP* to allow for M distributed agents to base their decision on local observations and interact in a partially observable environment [20]. In this framework, an agent might only reason about its own local state or observation. Also, his view of the environment might be different of the one of the other agents. As a result, the agents might be uncertain about the exact consequence that one of their actions might have on the environment [7].

When learning an optimal joint policy, blindly applying a greedy policy based on an estimated value function might not work. In a MAS, multiple joint actions may be optimal and as a result different agents might decide on different ways to break out the ties and the resulting joint action may be sub-optimal [5]. This is why some sort of inter-agent coordination of the individual learners is needed [21].

Similarly to the single agent case, the agent's goal must be reflected by the reward signal. If every agent in a MAS receives a global reward, the task is considered collaborative. A shared reward pushes the agents to seek maximisation of the common discounted return [5]. Also, it discourages laziness but might not provide a direct incentive to collaborate . As a downside, it does not scale well for complex tasks. Indeed, the reward signal might not provide enough feedback to their own action [12]. Learners involved in MARL can be divided in two main classes. *Joint-action learners* which are able to observe other agents actions and rewards and *independent learners* which consider the other agents as a part of the environment [6].

Finding/learning the optimal joint policy can either be done using *team learning* where a single learner is in charge of the team behaviour or by using *concurrent learning* where multiple learners try to improve parts of the team (each agent might have it's own learner) [12].

In team learning, although the state space might be large, single-agent learning techniques can be leveraged. With this approach the learning algorithm is centralised. If specialists are needed in the team, *heterogeneous team learning* should be used to obtain more diversity to the cost of a bigger search space. If all agents can have the same behaviour *homogeneous team learning* should be preferred. Despite having the same behaviour, they might have different execution properties as they might not have the same observation [12]. With concurrent learners, the problem is broken down in smaller "chunks" [22]. It has been shown that this method might be preferable if the agents need to concentrate on independent sub-problems. This method comes with the caveat that changes in the behaviour of an agent might ruin the learned behaviour of other agents by making their policy obsolete [12].

The simultaneous learning of autonomous agents makes the environment highly dynamic and non-deterministic. This has the aftereffect of making each agent pursue a moving target [4, 15]. As a result, with independent learning and the presence of multiple concurrent learners, the environment becomes non-stationary from a single agent's perspective [4, 12]. Indeed, the evolution of the state transition probability now depends on time as well as other agent actions and history. In fact, independent learning reduces the learning problem to a single-agent learning problem. In this setting, the agents mutually ignore each other and any notion of cooperation or coordination among agents is ignored [15]. Therefore, a direct or indirect way to coordinate needs to be provided to the agent[5].

Inter-agent communication is a possible way to mitigate the partial observability and the need to coordinate. Communication in MARL corresponds to altering the state or observation such that other agents can perceive the modification and decode information from it. If the communication channel is unrestricted, an agent could transmit all his available information. By doing so, the problem would be equivalent to a single agent system with full observability. As a result, some restrictions must be imposed. Such constraints can be the cost or the latency of the communication, the available vocabulary, etc. The communication can either be direct or indirect. Direct communication assumes a mean of sharing information directly to the other agent. It can be done by relying on a hard coded or learned communication protocol. Agents can also communicate indirectly by using an implicit transfer of information. This can be achieved through modification of the environment. For example, by leaving a trace behind the agent [12].

2.3 Related Work

Whether it is in the field of linguistic, psychology or AI, communication between multiple entities has seen a wide variety of research. Some work concentrate on interaction using predefined or learned protocols while others concentrate on the learning technique (evolutionary, reinforcement learning, etc). Recent works have also looked at the grounding and compositionality of the learned protocol.

One of the related research topics to this project is linked to referential games. Referential games are used to look at the pragmatic inference capabilities of learning agents in interactive reasoning tasks. A referential game widely used in the learning-to-communicate literature consists of having one agent guess what the other agent sees. For example, from a collection of images a target is selected. One of two agent sees the target and decides on a message to send. Based on the received message, the task of the second agent is to select the target from a set of images containing the target image as well as distracting images. [23] and [24] show that compositionality of a language learned in a referential game can be hampered by the type of observation used. Compositionality of a language is highly important. Indeed, the ability to construct complex expressions whose meaning is based on the meaning of its constituents can enable agents to communicate theoretically infinite expressions using only a finite dictionary.

In [23] and [24], the messages are constrained to a fixed length. [25] relaxes the fixed length messages by using sequences of discrete tokens with a variable length encoded using an LSTM. They demonstrate that in this setting, an order in the developed language emerges, i.e. the order in which the token are received is important. They also show that hierarchical encoding scheme and paraphrases can appear. [23] and [24] rely on REINFORCE for the training while [25] relax the discrete communication channel by using the Gumbel-Softmax estimator enabling them to train end-to-end as the problem is now fully differentiable.

Instead of referential games, the community has also looked at tasks with a longer horizon than a single time step. [26] proposed an environment where goals are private to each agent, but may involve other agents. For example, agent i may want agent j to do a specific action. In this situation, communication is required to coordinate because agent j can not observe agent i's goal. Using the Gumbel-Softmax estimator and end-to-end training, they show that agents can come up with an abstract language from experience only. [27] used the same environment to propose an alternative reinforcement learning training method using Actor-Critic to solve the task. [28] concentrate on a pursuit problem where two hunters need to coordinate to capture a prey. They show that communication can help obtaining better policies with the trade-off of slowing down the learning due to an increase in the number of rules to be held by the agents.

[29] proposed two approaches relying on Deep Q-Learning for solving riddled and multi-agent problems with partial observability and show that either of them can be used to learn communication protocol. Finally, [30] presents result showing that even though most learning communication might be effective, they are most of the time not interpretable or compositional.

Chapter 3

Experiments

3.1 Experimental Setup

As depicted in figure 3.1, three robotic arms are used for the experiment. The global task is to transport all cubes available at buffers 1 (B_1) and 2 (B_2) into buffer 3 (B_3) with the least amount of loss possible. When a cube reaches B_3 it is considered *sinked*. If it is lost in transit, it is considered *dropped*. Drop of cubes happens in two different situations. First, a buffer already containing the maximum number of cubes (full buffer) will drop any newly added element. The second way to lose a cube is related to the way a conveyor operates. The conveyor automatically transfers cubes from its head buffer to its tail buffer in a predetermined number of time steps. It does so by moving the head buffer to the other end of the conveyor. This makes the head buffer unavailable for deposit during the transfer process. During that period of time any attempt of depositing a cube on to the conveyor will result in a dropped cube. See appendix B for an in-depth look at the architecture used.

The robotic arms can either be scripted or managed by an agent while the rest of the entities reacts automatically to the state of the environment. The number of time steps required by the conveyors to move a cube from the head to the tail buffer is set to 1. All buffers related to the conveyors have a maximum size of 1 while the other buffers have no maximum size.

During the simulation and at each time step, the agents controlling the arms must select one action from their respective action space. At minimum, they all have access to the state information relative to the entities in their "circle of vision" as well as their own internal state to make their decision. The circle of vision is depicted in figure 3.1 by the circle around each arm. $agent_{arm1}$ has guaranteed access to the state of B_1, HBC_1 as

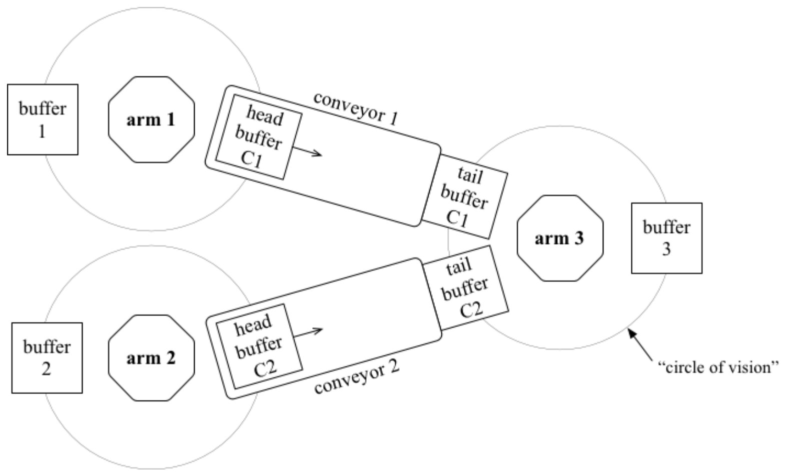


FIGURE 3.1: Schematic overview of the environment used. The figure depicts the entities involved and their relationship. The circle around the arms is referred as "the circle of vision".

well as the position of the conveyor 1. Indeed, the position of the conveyor is required to know the availability status of the head buffer. The observation space of $agent_{arm2}$ is similar. $agent_{arm3}$'s observation space contains the state of TBC_1 and TBC_2 as well as the state of B_3.

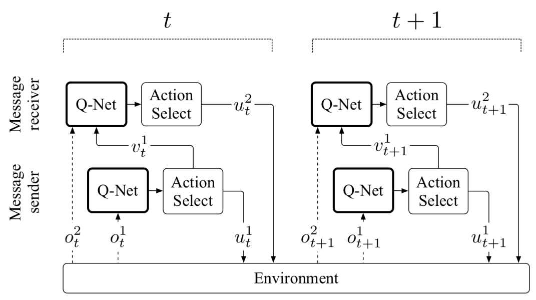
It is assumed that all agents have access to a *noop* action indicating that the agent does not wish to execute anything. When communication is enabled, $agent_{arm3}$ is the only agent that need to take both physical actions and communication actions. Its physical action space is composed of (1) move cube from TBC_1 to B_3, (2) move cube from TBC_2 to B_3 and (3) *noop*. The action space of the feeding arm_i for $i \in 1, 2$ is composed of (1) move cube from B_i to HBC_i and (2) *noop*.

arm_3 (*receiving* arm) is slower compared to the *feeding* arm arm_1 and arm_2. It takes the receiving arm 3 time steps to move a cube from a point to another while for the two feeding arm, a single time step is needed. This speed disparity removes possible intrinsic coordination between the feeding and receiving arms.

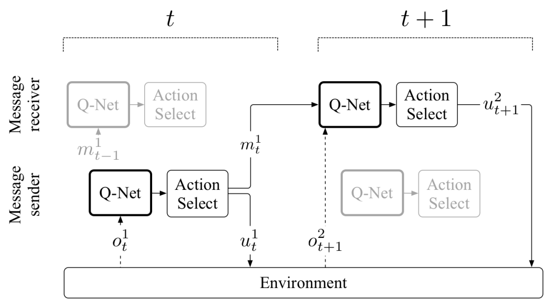
The environment will move from t to $t + 1$ only after all actions selected at time t have been applied to the environment. The execution of a selected action might be refused. This happens if the underlying physical arm is still executing a past action. For example, at t_0, the agent responsible for arm_3 decides to take action a (corresponding to moving a cube from p_1 to d_1). Because the speed of transport for arm_3 is 3, the underlying arm will not be responsive to any new action during t_1, t_2 and t_3. This means that the only action the agent should select during those time steps is the *noop* action.

3.2 Communication schemes

Communication can be approached from two different stand points. At each time step, beside a physical action, a communicating agent will also select a communication symbol (or message) $v \subset V \in \mathbb{R}^{K\times 1}$. The symbols are discrete elements taken from a vocabulary V of size K. The selected message v will then be transmitted to the other agents. It is assumed that the messages in the vocabulary are distinct. Then, based on both their individual observation and the messages received, the agents will be able to select a physical action. The used communication schema influences the time between the sending of a message and its reception by the other agents. The first approach inspired by [26] uses an instantaneous channel. A message selected at time step t is directly available to the other agents. On the other hand, communication can be delayed as described by [29]. With this approach, a message selected at time step t will only be available to the other agent at minimum at time step $t+1$. It is important to stretch that no prior signification is assigned to the vocabulary symbols. Meaning will be assigned during training time by the agents themselves. Figure 3.2 illustrates the differences between the two communication approaches.



Instantaneous communication The message sent by a communicating agent are directly available to the other agents for their physical action selection.



Delayed communication The message sent are available to the other agent at the following time step

FIGURE 3.2: Comparison of process flow between instantaneous and delayed communication [1]

In the experiments discussed in chapter 4, only $agent_{arm3}$ can communicate in a broadcast fashion to $agent_{arm1}$ and $agent_{arm2}$. Following the work of [29], $agent_{arm3}$ is separated into two sub-agents. One agent is used to select the communication action ($agent_{arm3c}$) and the other one to select the physical action ($agent_{arm3u}$). Both sub-agents share the same observation space.

[1]Figures are inspired by [29].

3.3 Reward function

The most straight forward reward would be related to the solving of the task. Unfortunately, initial experiments showed that this approach does not provide enough feedback to the agents to learn effectively. Instead, reward shaping is used. In order for the learner to strive for cooperative behaviour the reward signal is shared between the agents. The global reward is as follows:

$$r_t = \sum_{a \in A_t} R(s_{t-1}, a, s_t) + r_{time} \tag{3.1}$$

The global reward is hand-crafted from three sub-parts. As an incentive to avoid drop of cubes, a **cube reward** is added per cube transported during the time-step. A reward of -1 is given if the cube is dropped, 1 otherwise. A selected action might give rise to a **behaviour reward** of -1 if it was not realised by the underlying arm. This signal is used to avoid flooding of the meta controller by the learners with actions that can not be fulfilled. For example, this situation can happen when the underlying arm is still moving. An additional fixed reward (r_{time}) is added to he global reward for every elapsed time-step. A negative **time reward** can be used to push agents to be active and finish the task as soon as possible.

$$R(s_{t-1}, a, s_t) = c(s_{t-1}, a, s_t) + b(s_{t-1}, a, s_t)$$

$$c(s_{t-1}, a, s_t) = \begin{cases} -1 & : \text{action } a \text{ resulted in a drop of cube} \\ +1 & : \text{action } a \text{ resulted in a correctly transported cube} \\ 0 & : \text{otherwise} \end{cases}$$

$$b(s_{t-1}, a, s_t) = \begin{cases} -1 & : \text{action } a \text{ was not executed (refused)} \\ 0 & : \text{otherwise} \end{cases}$$

On top of the shared reward, a individual **communication reward** (r_{com}) can be added. The communication reward is related to usage cost of the channel. It might be positive to favour usage of the channel or negative to favour communication scarcity. It is assumed that the vocabulary at the disposal of a communicating agent contains an equivalent to "no message sent" which incurs no cost. This enables the learner to decide if sending a message is worth the cost.

3.4 Training

One training session is composed of 6000 episodes. An episode corresponds to one transport session happening in the environment described above. Before each episode, the world is reset. All cubes still present are removed and all robotic arms are reset. Afterwards, each buffer is filled with a random number of cubes. The number of cube put into each buffer follows a normal distribution $\mathcal{N}(\mu = 0.75 * \mathit{max\ buffer\ size}, \sigma = 3)$. An episode is considered finished either when a predetermined number of time steps has been done or when no cube are left to transport (all cube have either reached buffer 3 or have been dropped). Chapter 4 presents a comparison between learning scenarios as well as the influence of the different hyper parameters on the learned solutions. In order to evaluate the efficacy of the policies learned, the resulting learners are applied to 500 new random problems. The learning process is repeated 50 times. A solution is considered optimal if during the 500 evaluation problems 100% of the cube to be transported reach buffer 3 (B_3).

Centralised Learning

This approach uses a single agent making decisions based on a concatenation of all observation and action space of all agents. With this approach there is no autonomy of agents and the multi-agent task is transformed in a single agent task. The centralised learning is used as one of the baselines.

Decentralised Learning

When decentralised learning is used, each arm is managed by an autonomous agent learning at the same time in the same task. Both decentralised learning with full and partial observability are explored. In order to circumvent the inability of the agents to have a direct insight about the other agent behaviours in the partial observability case, communication is used.

3.4.1 Action selection and exploration

Learners are all implemented using vanilla tabular Q-learning with optimistic initialisation. The discount factor used during learning will be indicated for each experiment. The actions are selected to be the maximum estimated Q-Value relative to a given observation, $a = \arg\max_a q(o, a)$. In order to maintain exploration during training, an

ϵ-greedy policy is used with an exploration rate decreasing linearly over time from 0.5 to 0.05.

Chapter 4

Results & Discussion

In order to add stochasticity to the learning, $arm3$ is controlled by a scripted agent. When $agent_{arm3}$ has the ability to transport a cube, it will do so with 50% chance.

4.1 Learning approach comparison

Table 4.1 demonstrates the veracity of the assumption made in the introduction. The best hyperparameters are used for the training. The best hyperparameters are selected by looking at which parameters maximise $\frac{\textit{ratio of sinked cube}}{\textit{ratio of dropped cube}}$. The simulated problem can be solved optimally when the agents have full observability in a reliable manner. The learning can then be done in a centralised (CFO) or decentralised (DFO) manner. Removing the agent's access to the full observability of the environment makes the problem unsolvable (DLO). Figure 4.1 also shows that with DLO learning, the agents have a tendency to transport cubes even though they do not have certainty about the outcome of their action. This behaviour is studied in-depth in section 4.2. Also, table 4.1 confirms that introducing instantaneous communication can help mitigate the partial observability problem ($DLOCI$) and help solving optimally the problem.

Training method		Optimality
Centralised	CFO	1.00
Dec. Full obs.	DFO	1.00
Dec. Limited obs.	DLO	0.00
Dec. Limited obs. with delayed com. ($\lvert V\rvert = 2$)	$DLOCD$	0.00
Dec. Limited obs. with instantaneous com. ($\lvert V\rvert = 2$)	$DLOCI$	0.82

Table 4.1: Optimality ratio relative to the training method

Method	Optimality
CFO	1.00
DFO	1.00
DLO	0.00
DLOCI	0.82

$TBC1$ state	$TBC2$ state	Intent enc.
Emtpy	Empty	(11)
Emtpy	Full	(10)
Full	Emtpy	(01)
Full	Full	(00)

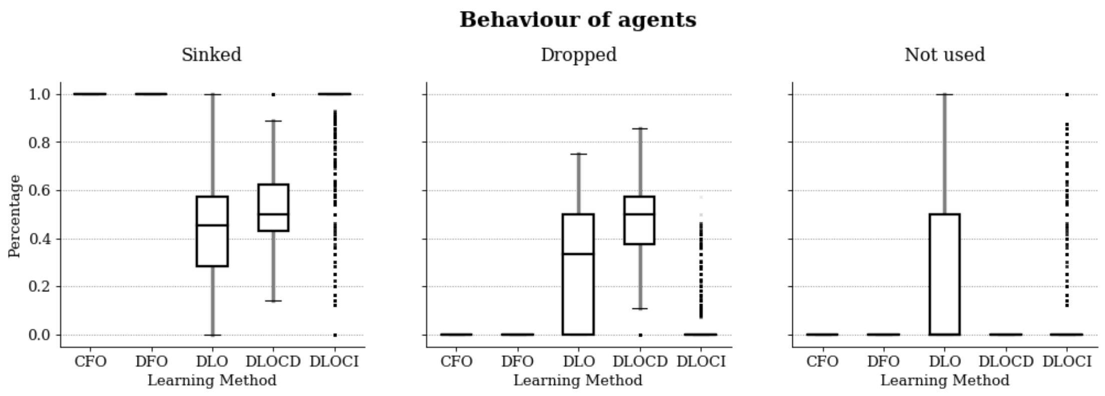


FIGURE 4.1: Behaviour of the agents relative to the training method. For each method, the best hyper parameters are used[1].

Interestingly, in the simulated problems, having a delayed communication ($DLOCD$) precludes the learning of optimal solutions. This might be explained by the absence of a direct link between communication and resulting action. With a delayed communication, a received reward at time t might be related to a communication made at time $t-1$. As a result, it is more difficult for the agents to agree on a grounding of the communication. [29] relies on centralised learning and decentralised execution to mitigate this problem. This approach to MARL being out of the scope of this work, instantaneous communication is used in the following analysis.

In order for the agent to have an incentive to transport cubes and to take into account their past action, a time-reward of -0.1 and a discount factor of 0.9 will be used as default values in the subsequent parts of the discussion.

[1]Figure 4.1 is constructed from 50 training session each composed of 500 evaluation problems. The reparation of cube usage observed in each problem is then aggregated and transcribed in the figure. Hyper parameters used: *CFO* ($\gamma = 0.9$, $r_{time} = -0.1$), *DFO* ($\gamma = 0.9$, $r_{time} = -0.1$), *DLO* ($\gamma = 0.9$, $r_{time} = 0.0$, $r_{com} = 0.0$), *DLOCD* ($\gamma = 0.9$, $r_{time} = -0.1$, $r_{com} = -0.1$), *DLOCI* ($\gamma = 0.9$, $r_{time} = 0.0$, $r_{com} = 0.0$).

4.2 Risk aversion

When communication is not available and control is decentralised, agents must take action with a much higher degree of uncertainty. The uncertainty raises from the absence of state information about the tail buffer of the conveyors for $agent_{arm1}$ and $agent_{arm2}$. In the *DLO* scenario, no optimal solutions are found and a certain number of solutions results in drop of cubes. As reflected in figure 4.1, they still have a tendency to transport cubes without the guarantee of a positive outcome.

In a specific training session, an agent is considered to be *risk averse* if it does not perform any other action than *noop* and *risk seeking* otherwise. The risk aversion of an agent can be seen as a measure of the trade -off between selecting an action with an unknown payoff compared to another one with a possibly lower, but more predictable payoff.

The *DLO* setting is particularly suited to study the risk aversion of the agents. Indeed, in this setting both $agent_{arm1}$ and $agent_{arm2}$ have no opportunity to coordinate with respect to the state of the tail buffer of the conveyors. Figures 4.2 and 4.3 show the average risk aversion. The risk aversion is calculated by looking at the learned behaviour of both $agent_{arm1}$ and $agent_{arm2}$. The risk aversion represents the ratio of solutions where both agents are risk averse, i.e do not transport any cubes. A risk aversion of 1 means that in 100% of the training sessions, both $agent_{arm1}$ and $agent_{arm2}$ learned to not transport cubes. On the other hand, a risk aversion of 0 indicates that in all the training sessions both agents were risk seeking and learned to send cube under uncertainty.

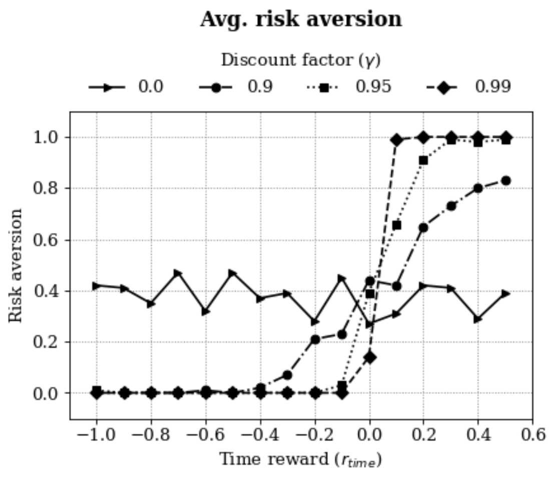


FIGURE 4.2: Average risk aversion relative to the time-reward.

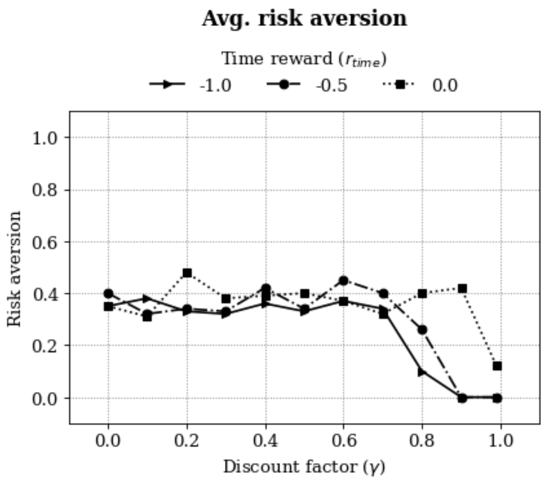


FIGURE 4.3: Average risk aversion relative to the discount factor.

Acting as a sanity check, when both the time reward and the discount factor are bigger than 0 the agents have a high risk aversion. The positive time reward pushes the arm to "run the clock". By not transporting any cube, they wait until the maximum number of time steps is reached and as a result maximise the long term return. On the other hand, as long as the agent's horizon is not too long, the time reward does not seem to influence the risk aversion (fig. 4.3). At the extreme, the discount factor of 0 the time reward only shifts all Q estimates by the same amount. As the discount factor gets closer to 1, the arms become more risk seeking. With the goal of maximising the long term reward, any transported cube will make the episode finish faster. In addition to shortening the episode, a few cubes might be transported correctly which increases the total long term reward.

4.3 Communication Analysis

4.3.1 Influence of communication reward

The cost related to the channel usage is introduced by adding a communication reward to the global reward for $agent_{arm3c}$. As explained in section 3.3, the communication reward is only added to the sender. As a reminder, the messages in the vocabulary from which the communication agent picks are labelled from 0 to $|V| - 1$. The usage of the message v_0 incurs no cost as it is the equivalent to "no message sent". In order to facilitate the analysis of the influence of the communication reward in the *DLOCI*, a vocabulary of size 2 is used.

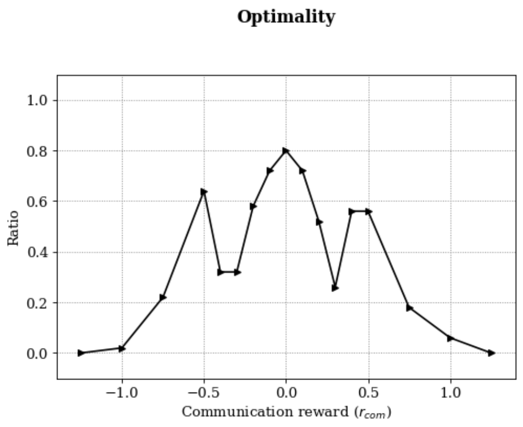


FIGURE 4.4: Optimality of learning when trained with hyperparameters $|V| = 2$, $r_{time} = -0.1$ and $\gamma = 0.9$ relative to the cost of communication.

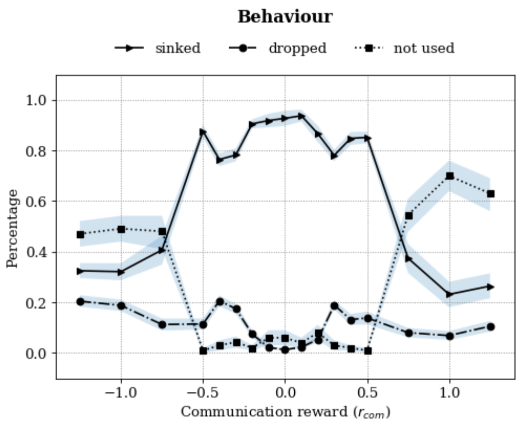


FIGURE 4.5: Behaviour of the agents when trained with hyperparameters $|V| = 2$, $r_{time} = -0.1$ and $\gamma = 0.9$ relative to the cost of communication.

Figures 4.4 and 4.5 shows that the ratio of optimal solution drops to 0 when the absolute value of the communication reward is too high. With such a communication reward, the agents reach a sub-optimal Nash equilibrium. $Agent_{arm3c}$ will learn to always send a specific message from the vocabulary depending on the sign of the communication reward. It will do so in order to maximise its own reward. Following convergence, an improvement in the solution can only be expected if all agents change their behaviour. One can see that with only two messages available the most reliable learning is done with a cost-less or an almost cost-less communication channel (i.e. $r_{com} \approx 0$). With no implicit constraint on the usage of the vocabulary there is a higher chance than the agents will converge to a optimal solution. The behaviour is similar for $|v| = 4$ when $t_{com} \leq 0$. When $t_{com} > 0$, the drop in optimality is not seen. Indeed, $agent_{arm3c}$ can now discard the use of v_0 and still have access to a sufficient vocabulary size to solve the problem optimally.

Grounding of communication

Grounding of communication between interacting entities is a widely researched topic. First introduced by Clark, Herbert H. and Brennan, Susan E. (1991) [31], it relates to the collection of "mutual knowledge, mutual beliefs, and mutual assumptions" that is required for successful communication between two entities. A common grounding criterion is that everyone involved has a clear enough understanding of the concept and intent transported through the communication channel to move forward correctly [32]. Also, in the setting of reinforcement learning, a link with the *new contribution* method proposed by [33] to reach grounding of communication can be made. This method relies on a partner moving forward with a new idea and waiting to see if its partners expresses confusion. In reinforcement learning, this can be seen as sending a specific message and looking at the resulting reward. The reward then acts as an indication of the confusion of the other agents [34].

$TBC1$ state	$TBC2$ state	Intent enc.	Who can transport safely a cube?
Emtpy	Empty	(11)	Both feeding arms
Emtpy	Full	(10)	Only $arm1$
Full	Emtpy	(01)	Only $arm2$
Full	Full	(00)	Not safe for both

TABLE 4.2: Abstracted sub states for $agent_{arm3c}$. The sub states represents the possible intents that can be transmitted though the communucation channel.

The grounding of communication focuses on the interconnection between the intent of the message sent and its interpretation by the receiving agents. Only protocols resulting in optimal solution will be studied. The protocols are inferred from the learned Q-Table. An optimal solution guarantees that at least one message is sent with the intent to indicate that it is not safe for both agents to transport a cube and interpreted correctly. It is important to note that even tough the intent of a message might be "safe for both", in reality $agent_{arm1}$ interprets this message as "safe for me" as it has no notion of existence of $agent_{arm2}$. The same can be said for the interpretation at $agent_{arm2}$. It is only when looking at the intent of the message and the reaction of both feeding arm to the message that it is possible to determine the overall grounding of the protocol.

The state space of $agent_{arm3c}$ is not limited to the status of the buffer $TBC1$ and $TBC2$. Consequently, the communicating agent could send more than a one symbol per sub state. However, it will have to do it at different time steps. The sub state can be seen as a categorisation on top of the state of $agent_{arm3c}$. For example, $agent_{arm3c}$ can be in sub state 00 while the underlying arm is moving or not which gives two different states. Also, $agent_{arm3c}$ can send the same message v in different sub states.

The grounding of the communication can be represented by a tripartite graph. Figure 4.6 depicts an example of such a graph. The nodes on the left side represent the intent of the message sent which is related to the sub states shown in table 4.2. The nodes on the right correspond to the interpretation of the message by $agent_{arm1}$ and $agent_{arm2}$. Both the intent and the interpretation are encoded using a binary representation. A "1" indicates "safe to transport" and a "0" indicates "not safe to transport". For example, "10" on the intent side means that it is safe for $agent_{arm1}$ to transport a cube while it is not for $agent_{arm2}$. On the other hand, "10" on the interpretation side means that $agent_{arm1}$ interprets the message as "safe for me to transport" while $agent_{arm2}$ interprets it as "not safe for me to transport". Finally, the nodes in the middle represent the vocabulary. The example presents a possible grounding of communication leading to an optimal solution when $|V| = 2$. This type of protocol only transmits two intents through the channel: the fact that both feeding arms can send safely as well as the opposite. Each intent correctly transmitted though the channel is represented by a coloured node on both side of the graph. A dashed arrow depicts an intent not correctly interpreted by at least one of the the feeding agents. When the number of intent correctly transmitted is equal to the number of available symbols, the protocol will be called ideal, otherwise non-ideal.

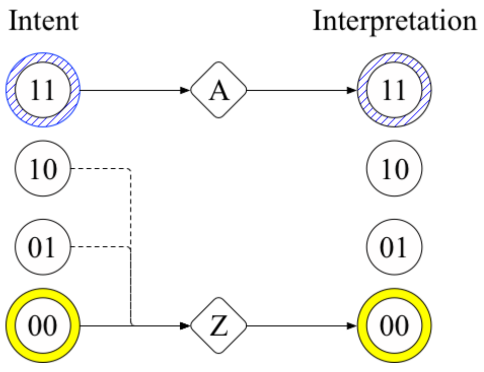


FIGURE 4.6: Example of an ideal protocol for $|V| = 2$ leading to an optimal solution.

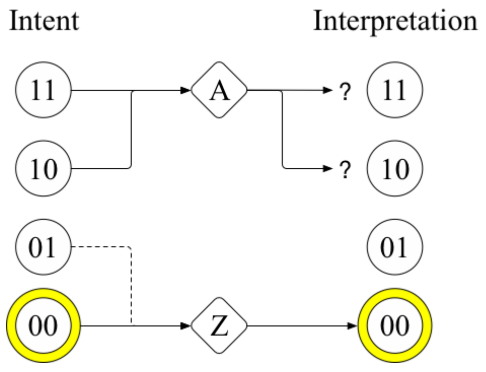


FIGURE 4.7: Example of protocol for $|V| = 2$ leading to a sub-optimal solution.

Out of the 16 possible types of protocol, only five types can lead to an optimal solution (see fig. 4.5 for the list). In order to distinguish them, a binary encoding of the intents correctly interpreted is used. Each digit of the binary strings of length 4 represents one of the intent present in table 4.2. If the related intent is correctly transmitted and interpreted, a 1 will be present in the encoding, otherwise 0 is used. The example above thus result in the encoding 1001. We distinguish between the protocol type and the protocol itself. A protocol type indicates which intent is correctly interpreted. On the other hand, a protocol is a realisation of a protocol type where the symbols of the vocabulary have been assigned to a specific vocabulary node in the graph.

It is clear that with a vocabulary of size 2 the only grounding leading to an optimal solution is based on the protocol type 1001. As a result, there exists only two valid protocols and both are ideal protocols. A valid protocol is a protocol leading to an optimal solution. They correspond to the two different assignments of the symbols of the vocabulary to the node A and Z. One where the symbol v_1 is assigned to node A and v_0 to node Z and the opposite. Figure 4.7 depicts an invalid protocol. Compared to before, $agent_{arm3c}$ also tries to convey information about the sub sate "safe for $arm1$ but not safe for $arm2$" via symbol A. Because of this change in the protocol, it is now impossible for $agent_{arm2}$ to decipher whether it is safe to transport a cube or not. For $agent_{arm2}$, symbol A now indicates that $TBC2$ can either be full or empty. As a result $agent_{arm2}$ can no longer definitively know the intent conveyed by A. In this situation, $agent_{arm2}$ can not know the state of $TBC2$ with certainty. As a result will either learn to not send any cube or send cube by becoming risk seeking. Either way, this will result into a non optimal solution with less than 100% of the cube sinked.

Influence of communication-reward on the grounding of communication

When looking at optimal solutions with a vocabulary of size 2 $V = (v_0, v_1)$ over a perfect channel, the communication reward influences the meaning that the agents agree to attribute to specific symbols (fig. 4.8). When the cost of using the channel is greater than 0, the communicating agent has an incentive to use the costly message (v_1) when there are no possibilities of receiving any other positive reward. As a result, v_1 will be used to indicate the intent "00" (i.e. "not safe to transport for both"). On the other hand, when the cost is lower than 0, the inverse happens. The agents will converge to a grounding of the communication using the costly message (v_1) to indicate that both feeding arms can transport cube safely. The positive reward received after a successful transport is used to compensate the usage cost of the channel. When there is no cost in using either word of the vocabulary, no preference in the grounding of the protocol is seen. Half the time the costly symbol will convey information that will push the feeding agents to transport and the other half will indicate they should not move.

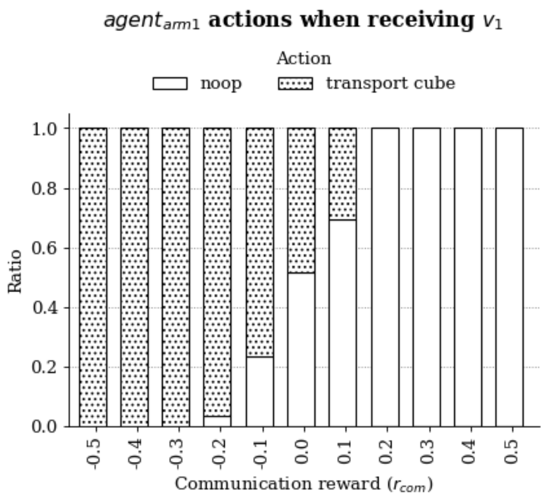


FIGURE 4.8: $agent_{arm1}$ reaction to receiving message v_1. Only looks at state where $agent_{arm1}$ could potentially transport a cube.

4.3.2 Vocabulary size

Until now, only solutions with the minimum amount of symbols necessary to solve the task were considered. As shown before, not all possible protocol graphs and assignments of the vocabulary can lead to an optimal solution and considered valid. The number of valid protocols relative to the size of the vocabulary is shown in table 4.3.

$\lvert V \rvert$	Valid assignments	
2	12.5%	(2/16)
3	37.5%	(24/64)
4	57.8%	(146/256)
8	89.2%	(58466/65536)

TABLE 4.3: Number of valid vocabulary assignments relative to the size of the vocabulary. Representation: (valid protocols/ possible protocols)

.

$\lvert V \rvert$	optimality
2.0	0.78
3.0	1.00
4.0	0.94
8.0	0.96

TABLE 4.4: Optimality ratio for 50 training session with hyperparameters $r_{time} = -0.1$, $r_{com} = -0.1$ and $\gamma = 0.9$

As shown in table 4.4, the increase in vocabulary size also brings an increase in the probability of getting an optimal solution at the end of a training session. With a bigger vocabulary there is a higher chance to converge to an optimal solution compared to $|V| = 2$. For $|V| = 2$, a single misinterpretation will make the solution sub-optimal.

When more than two words are available, most valid protocols can be simplified. Even tough the agents could convey more than two intents through the channel, it is not required to solve the problem optimally. In case the sender is not precise enough or the receivers are not able to learn correctly the intent of each symbol, it might be possible to discard the intent or the symbol altogether.

Complexity

As representation of the complexity of a learned protocol, the number of intents that are correctly interpreted is used. As a result, in order to achieve the maximum complexity the vocabulary needs to be composed of a minimum of 4 symbols. Table 4.5 shows the different types of protocol and their respective ratio out of the pool of valid protocols [2].

[2]The assignments are generated using random sampling of possible protocols. Each generated protocol is then checked to see if it can lead to an optimal solution. If it is the case, the type of protocol is then determined.

Protocol type	$\lvert V\rvert = 2$	$\lvert V\rvert = 3$	$\lvert V\rvert = 4$	$\lvert V\rvert = 8$
0111	0	<u>25%</u>	24.6%	9.9%
1001	<u>100%</u>	25%	9.5%	9.9%
1011	0	<u>25%</u>	24.6%	4.3%
1101	0	<u>25%</u>	24.6%	9.9%
1111	0	0	<u>16.4%</u>	<u>69.8%</u>

TABLE 4.5: Distribution of the type of protocols relative to the vocabulary size. For each vocabulary size, ratios relative to ideal protocols are underlined.

Figure 4.9 shows that even though the vocabulary size increases, it is not guaranteed that the learned protocol will be complex or ideal. Also, the agents do not ground the learned protocols uniformly. In most of the training sessions, the intent conveyed by the learned protocol is limited to 01, 10 and 11 (protocol 0111).

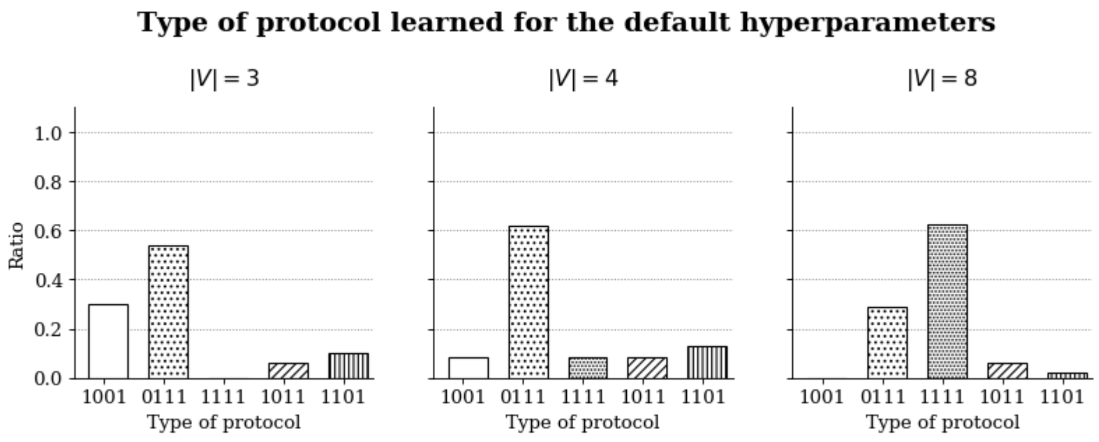


FIGURE 4.9: Empirical distribution of protocol types for 50 training sessions with hyperparameter $r_{time} = -0.1$, $r_{com} = -0.1$, $\gamma = 0.9$

The protocols have thus a tendency to only convey individual safety intent except when the vocabulary size is greater than the number of possible intents. In this setting, the protocol is more often ideal (indicates all intents, i.e. protocol type 1111). This behaviour can probably be attributed to the fact that the protocol 1111 is the most common assignment for $|V| = 8$.

An example of a learned protocol with a non-ideal usage of a vocabulary of size 4 is depicted in 4.10. In this example $agent_{arm3c}$ tries to convey to two intents via symbol A. Contrary to $agent_{arm2}$, $agent_{arm1}$ can correctly interpret symbol A. For $agent_{arm1}$, the symbol A always indicate that it is safe to transport a cube. On the other hand, symbol A conveys two contradictory intents to $agent_{arm2}$ and as a result the agent will not be able to leverage the received communication for its decision process. An ideal protocol with 4 messages can be found in figure 4.11. With such protocol, all messages indicate a different sub state and the vocabulary is maximally used.

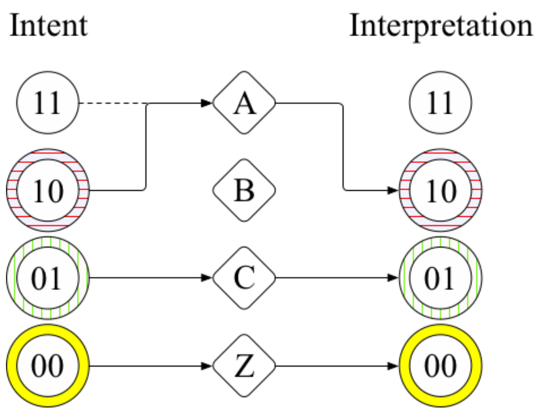


FIGURE 4.10: Non-ideal communication protocol for $|V| = 4$ (protocol type : 0111)

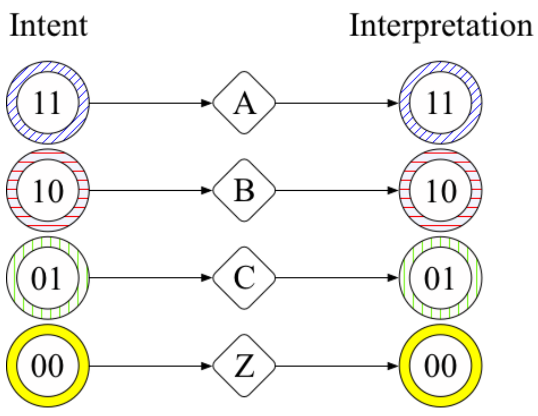


FIGURE 4.11: Ideal communication protocol for $|V| = 4$ (protocol type : 1111)

Influence of the discount factor on complexity

The ratio of solutions with an ideal protocol (fig. 4.12) can be increased by lowering the discount factor. By being myopic, the goal is to receive a positive reward as often as possible. In our experiments, a positive reward is received each time a cube is transported correctly. As a result, a low discount rate pushes $agent_{arm3c}$ to indicate as soon as possible an empty buffer. Also, the grounding of communication is facilitated as the estimated Q function contains less noise coming from decisions made in different states. At the extreme, with a discount rate of 0, the agents receive a direct feedback about the quality of the communication. This increase is visible for all vocabulary size as shown in figure 4.13.

4.4 Unreliable channel

Up to now, the channel has been assumed to be fully reliable. As a result not sending a message can be seen as a standalone word of the vocabulary. Moving to an unreliable channel which can drop messages introduces a prerogative on the usage of the channel. The message v_0 ("no message sent") must now be grounded as intent "00" (i.e "not safe for both"). Indeed, if any of the feeding agent ground the communication differently, a dropped message might be misinterpreted and lead to a drop of cube. This behaviour is reflected in figure 4.14.

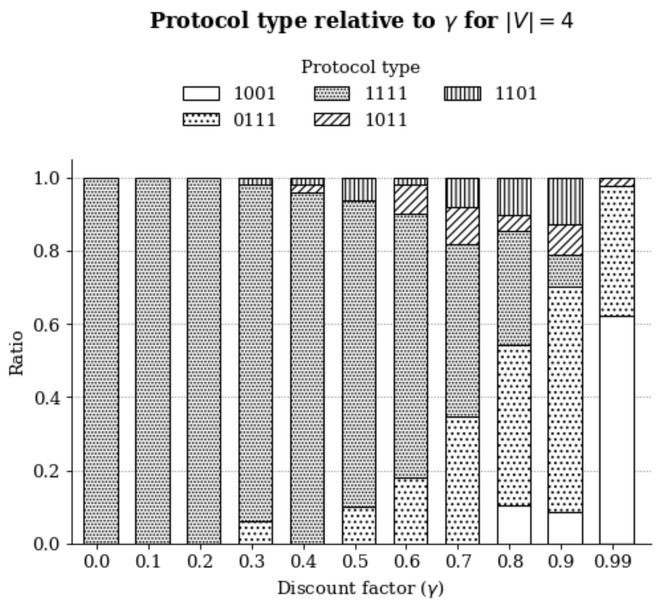


FIGURE 4.12: Distribution of protocol types relative to the discount rate for $|v| = 4$.

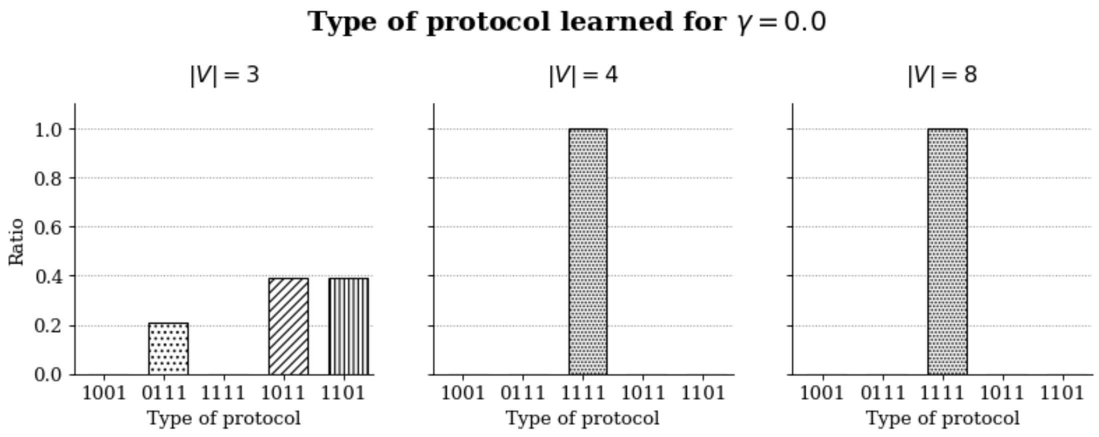


FIGURE 4.13: Empirical distribution of the protocol type for 50 training sessions with hyperparameter $r_{time} = -0.1$, $r_{com} = -0.1$, $\gamma = 0.0$

As shown in table 4.6, by putting a prerequisite usage of message 0, the number of possible valid protocols is reduced. As expected, due to this reduction, the number of optimal solutions is lower. Also, no optimal solutions are found when the drop rate is too high (fig 4.15).

$\|V\|$	Valid assignment	
2	6.2%	(1/16)
3	14.0	(9/64)
4	19.1%	(49/256)
8	27.5%	(16129/58466)

TABLE 4.6: Valid vocabulary assignments when channel is unreliable. This channel requires the v_0 to convey intent 00.

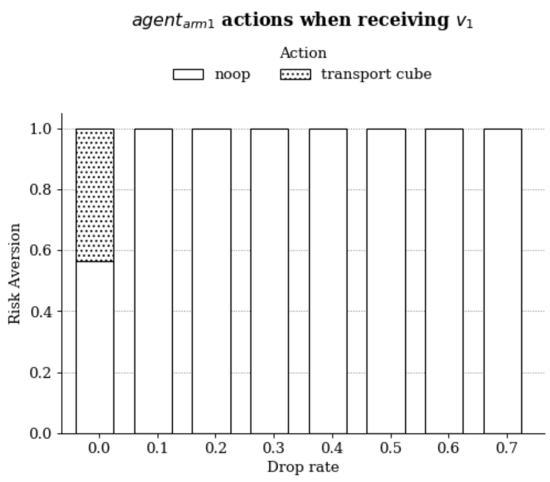


FIGURE 4.14: Reaction to received communication relative to the drop rate.

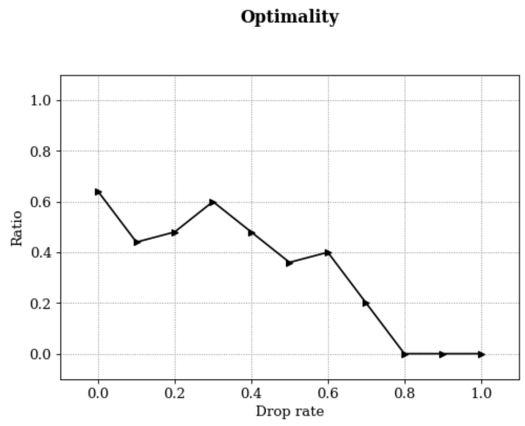


FIGURE 4.15: Optimality of learning relative to the drop rate.

4.5 Fully agent controlled

Finally, the constraint that the agent controlling $arm3$ is scripted can be relaxed. Table 4.7 shows that the scenario used can also be solved using fully decentralised learning with each arm being controlled by a learning agent.

Measurement	$\lvert V \rvert = 2$	$\lvert V \rvert = 4$
Optimality	0.86	0.96
Sinked (%)	96.37 ± 12.48	98.83 ± 7.80
Dropped (%)	1.62 ± 6.84	7.66 ± 1.01
Not used (%)	2.01 ± 10.75	1.59 ± 0.15

TABLE 4.7: Optimality and behaviour when control is leanrned in a fully decentralised fashion.

Chapter 5

Future Work and Conclusion

5.1 Future research

Several lines of research arising from this work should be pursued. Firstly, investigating more complex problems and learning methods might prove important. The current work relies solely on tabular Q-Learning and might be too limited for learning more complex tasks. Also, future research is needed to investigate the possibilities of using delayed instead of instantaneous communication. Finally, in the current work, the agents have no implicit incentive to be fast. It is thus possible to assume that the influence of the discount rate on the quality of the protocols might change when the incentive exists. This assumption should be addressed in future studies.

5.2 Conclusion

This work demonstrates that a simulated transport problem involving robotic arms can be solved using decentralised learners and inter-agent communication. Being able to solve a reinforcement learning problem in a decentralised manner can help when the problem becomes intractable by a centralised scheme. Furthermore, this work shows that increasing the vocabulary size available to a communicating agent improves the quality of the learned solution. On a higher level, the cost of communication can be used to influence the grounding of communication used by the agents. When a problem can be aliased to a referential game, a low discount rate can be used to increase the chance of converging to an ideal protocol.

Appendix A

Determination of the grounding of the protocol

The grounding of the communication is required when the protocol needs to be analysed. The determination of the grounding is done in 3 steps. In the first one, the interpretation given to each symbol of the vocabulary is determined. Then, the intent conveyed by each symbol is looked at. Finally, the information gained by the two preceding steps is merged to get the final grounding. Before looking at the grounding of the communication, the learned agents must be evaluated. Only protocol leading to an optimal solution are of interest. Afterwards, the greedy policy is determined by looking at the Q-estimate of each agent.

The following presents how the grounding is found. The example is taken from a training sessions with parameters $r_{com} = -0.1$, $r_{time} = -0.1$, and $\gamma = 0.9$.

Interpretations of the symbols The interpretation of the symbols by the feeding arms $agent_{arm1}$ and $agent_{arm2}$ is looked at first. In order to determine the interpretation, it is sufficient to look at what action the agent takes relative to the message received. Thus, only the states in which the agents could potentially transport a cube are of interest. If the agent decides to transport a cube, the message has been interpreted as "safe for me" and "not safe for me" otherwise. It is then possible to construct a table of interpretation and the right part of the protocol graph (tab. A.1, fig. A.2).

Symbol	$agent_{arm1}$	$agent_{arm2}$
v_0	not safe	not safe
v_1	not safe	safe
v_2	not safe	not safe
v_3	safe	safe

TABLE A.1: Example of an interpretation of symbols by $agent_{arm1}$ and $agent_{arm2}$.

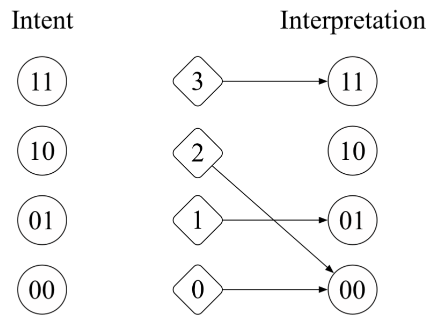


TABLE A.2: Protocol graph related to the interpretation of symbols by the feeding agents as shown in tab. A.1.

It is also possible to represent the assignment of the symbols using a set notation. This notation is useful when the grounding process needs to be implemented programmatically (tab. A.3).

interpretation-11	=	$\{v_3\}$
interpretation-10	=	$\{\emptyset\}$
interpretation-01	=	$\{v_1\}$
interpretation-00	=	$\{v_0, v_2\}$

TABLE A.3: Vocabulary assignment related to the interpretation.

intent-11	=	$\{v_1, v_3\}$
intent-10	=	$\{v_2, v_0\}$
intent-01	=	$\{v_1, v_2\}$
intent-00	=	$\{v_0, v_2\}$

TABLE A.4: Vocabulary assignment related to the intent.

Intents of the symbols The intent transmitted by each symbol is then looked at. For this, the Q-table of $agent_{arm3c}$ is used. By looking at each sub state (see table 4.2), the sets containing the symbols relative to each intent can be constructed. Continuing with the same example, the sets presented in table A.4 can be constructed. Based on this table, the protocol graph can then be completed (fig. A.1).

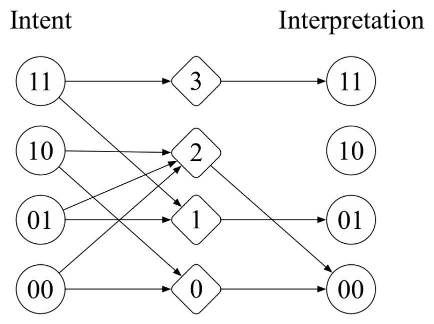


FIGURE A.1: Protocol graph related to the intent transmitted by the symbols by the receiving agent.

Grounding of the protocol Finally, based on the sets in tables A.3 and A.4, it is possible to determine the final grounding of the communication. Table A.5 shows the final results for the example. The constructed sets enable the creation of the protocol graph depicting which intent is correctly transmitted through the channel (fig. A.2).

grounding-11	=	intent-11 ∩ interpretation-11	$= \{v_3\}$
grounding-10	=	intent-10 ∩ interpretation-10	$= \{\emptyset\}$
grounding-01	=	intent-01 ∩ interpretation-01	$= \{v_1\}$
grounding-00	=	intent-00	$= \{v_0, v_2\}$

TABLE A.5: Vocabulary assignment related to the grounding.

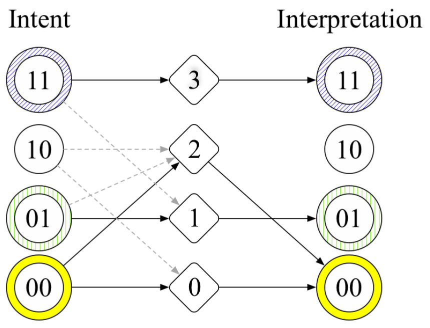


FIGURE A.2: Final protocol graph related to the example.

The protocol type of the example is thus 1011. This learned protocol is non-ideal even though it leads to an optimal solution.

Appendix B

Architecture overview

The learning environment used during the experiments is inspired by the OpenAI multi-agent environment [27]. The simulation can be separated into distinct parts. The *world* is the abstract concept where the agents will interact with each other. The second part of the architecture is involved into the reinforcement learning process. It is separated into *scenarios* and *runners*. Figure B.1 depicts the overall architecture. The scenarios act as an interface between the world and the runners while the latter is used to manage the necessary process for the reinforcement learning. In order to simulate the interaction between entities, discrete time steps are used. For clarity, the following assumes that cubes are being transported.

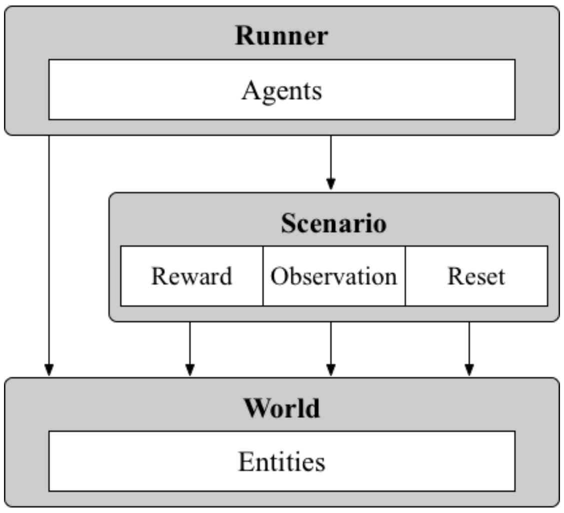


FIGURE B.1: Overview of the experiment architecture

B.1 World

The world is the sandbox in which all *entities* involved in an experiment will interact. Three different representations of physical objects were used. The internal state of all entities can be queried to enable the creation of the observation required by the agents. Entities can be separated into two classes. After being created, **scripted** entities will react to the changes in the environment automatically while **dynamic** entities are controlled by an external process. Dynamic entities are linked to a learning agent that will decide on the next action.

B.1.1 Buffer

The buffer can be seen as a wrapper around a FIFO queue. It is the equivalent to a crate where cubes can be stocked. The size of the buffer can be limited or infinite. When there is no space left in the queue, any new element added to it will be considered lost (i.e. dropped) for the remaining of the episode.

Available endpoints to other entities

(1) **get**: If a cube is available in the buffer, takes it out and returns it.

(2) **put**: If space is available in the queue, puts the given cube in the buffer. Drops it otherwise.

Available state information

- Buffer full status (boolean)
- Buffer empty status (boolean)
- Current size of the queue (integer)
- Maximum size of the queue (integer)

B.1.2 Conveyor

A conveyor acts as an automatic non-instantaneous transporter of cubes between two different points. In a nutshell, a cube placed at one end of the conveyor (*head*) will be available at the other end (*tail*) after a delay t. The head and tail of the conveyor are also buffers as described in section B.1.1.

The conveyor is automatic in the sense that as soon as its head buffer is full, it will start the transfer. It is assumed that the head buffer is moving along the conveyor making it unavailable for deposit during the transportation. After t time steps have elapsed, the content of the head buffer will have been transferred to the tail buffer. As for a normal buffer, any cube not fitting in the tail buffer will be dropped. Similarly to the buffer entity, two endpoints are available.

Available endpoints to other entities

(1) **get**: If a cube is available in the tail buffer, takes it out and returns it.

(2) **put**: If space is available in the head buffer and the conveyor is not moving, puts the given cube in the buffer. Drops it otherwise.

Available state information

- Speed t of the conveyor (integer)
- Position of the conveyor (integer)
- Buffer information for head buffer
- Buffer information for tail buffer

B.1.3 Robotic Arm

The interaction with a robotic arm is done via two different controllers depicted in figure B.2. The *physical controller* provide a low-level API. The available endpoints in the low-level API are abstractions of the atomic actions a robotic arm can do. Such action can be moving to point A, gripping, releasing, etc. for example. For a more in-depth look at the physical controller, see C. The low-level API is then leveraged by a *meta controller* to provide high-level actions endpoints. The meta controller combines low-level actions into higher-level actions such as transporting an object from point A to point B. In order to populate the high-level API, it is assumed that the possible pickup and deposit positions are fixed and known in advance.

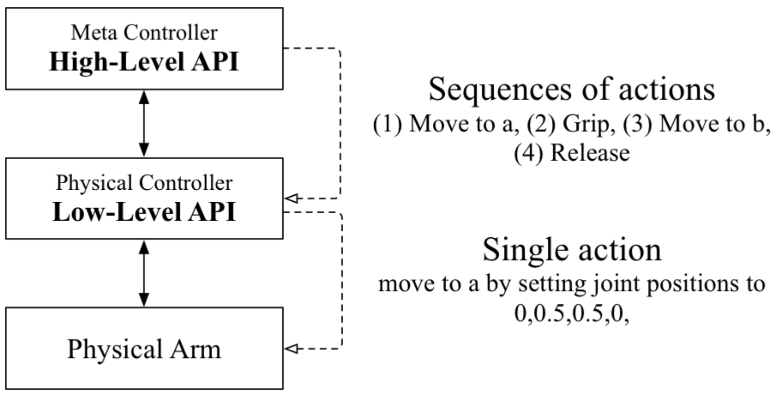


FIGURE B.2: Overview of the architecture used to control the robotic arms

Available endpoints to higher level

(1) $move_{p_i,d_j}$ provides endpoints to transport a cube from any pickup position p_i to any deposit position d_i.

(2) **noop** indicates to the physical controller that no action should be taken.

Available state information

- High-level action taken at the last time step.
- Busy status (indicate if an low-level action is being realised)
- Number of possible actions

B.2 Scenarios & Runners

A scenario provides a view of the underlying world specifically tailored to the need of a reinforcement learning experiment. It creates the reward and the observation vector for each agent involved. One other duty of the scenario is to reset the world into a predefined state.

Each training session is managed by a runner. The role of the runner is to orchestrate and manage the learning. It will provide the necessary observation to the agents for their decision and update process. The runner is also in charge of indicating to the world which action should be realised in the next time step.

B.2.1 Learning Cycle

The learning is separated into episodes. An episode can terminate either when the defined scenario is considered solved (example. no cube left to transport) or when a predefined number of steps have been done. The world is reset after each episode. The created framework is general and applies the following learning cycle.

Initialisation

(1) **Create the world** by creating the involved entities (buffers, conveyors, arms, etc.) as well as their relationship. Only done once per training session.

(2) **Reset the world** into an initial state (ready for training). This step is repeated before each episode.

Training

The training process is managed at the runner level. The current implementation assumes that all actions and all environment changes resulting from those actions have been performed before the world moves to the next time step.

(1) **Get observation**: Get $o_t^{a_i}$ for every agent a_i by querying the scenario.

(2) **Action selection**: Each agent selects his action based on $o_t^{a_i}$.

(3) **Apply selected action**: The selected actions are applied and all entities states are updated accordingly.

(4) **Get feedback from the environment**: Based on the outcome of the actions, reward signal and next observations are constructed by the scenario.

(5) If the task is considered solved (ex: no object left to transport in the system), the episode terminates. Otherwise go back to 1

Appendix C

Robotic arm Python wrapper

Due to time restriction, the interactions with the physical simulator or the real arms have not been implemented. Despite this, the outline of the physical controller is detailed below for future reference.

C.1 Robotic interface

The Robot Operation System (ROS) provides a software framework to help developers create software involving robots. It includes several open source libraries and tools (BSD license) to "encourage collaborative robotics software development" by providing useful abstraction over hardware, drivers, etc. Since its inception in 2007, ROS has been used to interface with a wide range of robots. Due to the collaborative approach taken by the ROS developers, hundreds of libraries as well as robotic interface have been contributed to the ROS eco-system [35]. This in turn enables developers to concentrate on innovation by building on each other's work.

ROS is a distributed framework of processes (referred as Nodes in the documentation) performing tasks. The framework is designed to be loosely coupled where each node is supposed to be responsible for a single task.

The communication between nodes is based on a publish-subscribe pattern using named logical channels called topics. Each node can then send message or receive message by either publishing or subscribing to a topic. ROS also provides services to cope with the need of remote procedure call (RPC) request / reply interactions. The communication protocol is defined by a pair of messages (request and reply). In order for the different node to find each other, ROS provides a Master nodes responsible for the registration as well as the look-up of topics [36]. After successful registration and look-up, the nodes

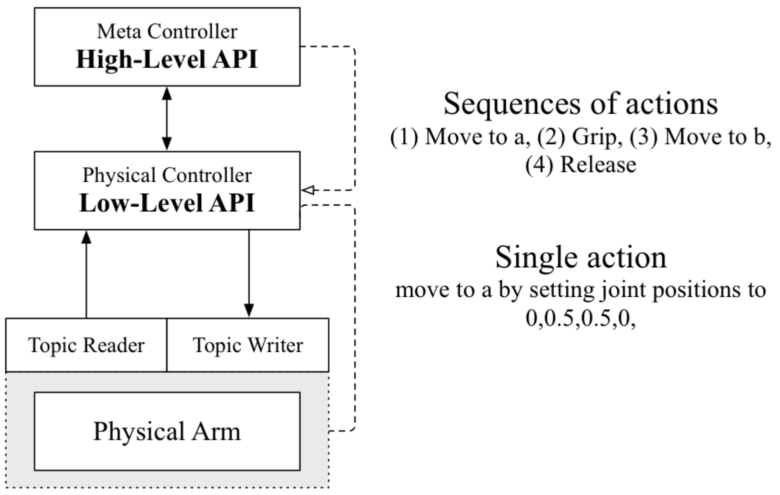


FIGURE C.1: Overview of the software architecture layers. The greyed items are only here for comprehension purposes and are not part of the physical controller.

communicate peer-to-peer by publishing a message to a topic or by sending a message to a service. A message is a simple data structure composed of typed fields.

Gazebo provides an open source robot simulation environment. It provides a wide range of tool to help software developers to easily design, test, improve algorithms or robots in realistic scenarios. Gazebo can directly and easily interface with ROS in place of the actual physical robot.

C.1.1 Robotic arm

The physical robotic arm selected was the PhantomX Pincher Robot Mark II [37] or commonly referred as the Turtlebot arm. This arm provides 4 degrees-of-freedom as well as pinching. A model of the arm is available both for ROS and Gazebo.

C.2 Architecture design

The goal of the physical controller is to abstract and encapsulate the possible action of the different robotic arms.

C.2.1 Overview

Figure C.1 shows the different level of abstraction implemented. The interface level provides two abstracted access points to ROS. A topic reader provides access to the latest message published in the topic enable the higher level to know the state of the arm while a topic writer enables the publication of a message to alter the state.

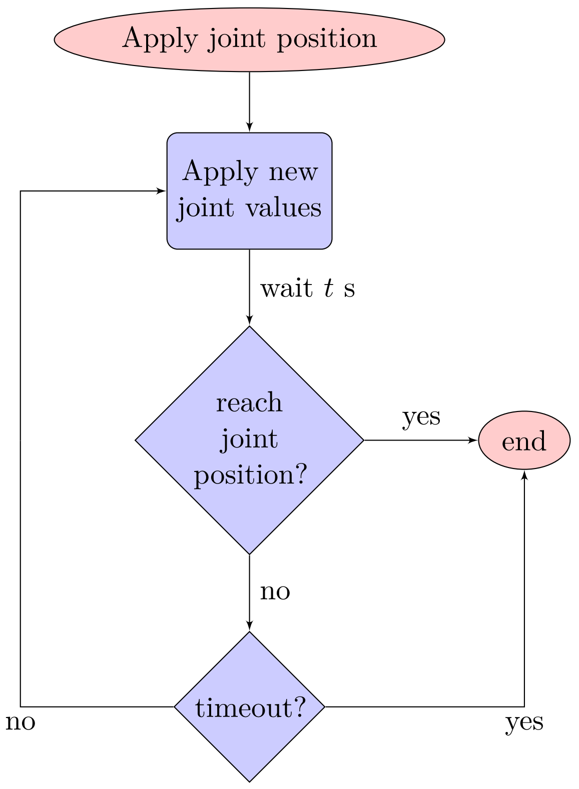


FIGURE C.2: Example of work-flow abstracted by the arm controller. For the user, after applying a new set of joint position, the controller will either return a zero or non-zero exit status depending on the result of the interaction with the "physical" arm. A non-zero exit code could be returned if a user breaks the arm by applying malicious values.

The two interfaces are used to create the physical controller. The goal of the controller is to provide a simple low-level API to interact with the underlying physical object (either real or simulated). The controller abstracts all the steps taken in order to execute an action or raise an exception in case of an error. Figure C.2 shows the abstracted process for applying new joint angles to the arm. Resting on this process and knowing the joint-position needed to reach point A, it is then possible to create a "move to point A" endpoints. The architecture leaves the error handling to the user.

Bibliography

1. Mnih, V. *et al.* Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs].* arXiv: 1312.5602. http://arxiv.org/abs/1312.5602 (2018) (Dec. 19, 2013).
2. Silver, D. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529,** 484–489 (Jan. 27, 2016).
3. Kaelbling, L. P., Littman, M. L. & Moore, A. W. Reinforcement Learning: A Survey. *arXiv:cs/9605103.* arXiv: cs/9605103. http://arxiv.org/abs/cs/9605103 (2018) (Apr. 30, 1996).
4. Busoniu, L., Babuska, R. & De Schutter, B. A Comprehensive Survey of Multiagent Reinforcement Learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* **38,** 156–172 (Apr. 1, 2008).
5. Busoniu, L., Babuska, R. & De Schutter, B. in *Studies in Computational Intelligence* 183–221 (July 17, 2010). doi:10.1007/978-3-642-14435-6_7.
6. L. Leottau, D., Ruiz-del-Solar, J. & Babuska, R. Decentralized Reinforcement Learning of Robot Behaviors. *Artificial Intelligence* **256.** doi:10.1016/j.artint.2017.12.001 (Dec. 1, 2017).
7. T. J. Spaan, M., J. Gordon, G. & Vlassis, N. *Decentralized planning under uncertainty for teams of communicating agents* in *ACM Transactions on Multimedia Computing, Communications, and Applications - TOMCCAP* **2006** (Jan. 1, 2006), 249–256. doi:10.1145/1160633.1160678.
8. Alpaydin, E. *Introduction to Machine Learning* 2nd. ISBN: 978-0-262-01243-0 (The MIT Press, 2010).
9. Sutton, R. & G. Barto, A. Reinforcement Learning: An Introduction. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* **9,** 1054 (Feb. 1, 1998).
10. Yves Glorennec, P. Reinforcement Learning: an Overview (Apr. 6, 2001).

11. G. BARTO, A. & Dietterich, T. 1 Reinforcement Learning and its Relationship to Supervised Learning. *Handbook of Learning and Approximate Dynamic Programming* (Dec. 3, 2008).

12. Panait, L. & Luke, S. Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems* **11,** 387–434 (Nov. 1, 2005).

13. Puterman, M. Markov Decision Processes : Discrete Stochastic Dynamic Programming / M.L. Puterman. *Technometrics* **37.** doi:10.2307/1269932 (Aug. 1, 1995).

14. Bellman, R. E. *Dynamic Programming* Google-Books-ID: fyVtp3EMxasC. 388 pp. ISBN: 978-0-486-42809-3 (Courier Corporation, 2003).

15. Bloembergen, D., Tuyls, K., Hennes, D. & Kaisers, M. Evolutionary Dynamics of Multi-Agent Learning: A Survey. *Journal of Artificial Intelligence Research* **53,** 659–697 (Aug. 1, 2015).

16. Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine Learning* **3,** 9–44. ISSN: 0885-6125, 1573-0565 (Aug. 1, 1988).

17. Watkins, C. J. C. H. & Dayan, P. Q-learning. *Machine Learning* **8,** 279–292. ISSN: 0885-6125, 1573-0565 (May 1992).

18. Lewis, D. *Convention: A Philosophical Study* (Wiley-Blackwell, 1969).

19. Altman, E. & Asingleutility, I. *Constrained Markov Decision Processes* (1999).

20. Bernstein, D. S., Zilberstein, S. & Immerman, N. The Complexity of Decentralized Control of Markov Decision Processes. *arXiv:1301.3836 [cs].* arXiv: 1301.3836. http://arxiv.org/abs/1301.3836 (2018) (Jan. 16, 2013).

21. Busoniu, L., De Schutter, B. & Babuska, R. *Decentralized Reinforcement Learning Control of a Robotic Manipulator* in (Jan. 8, 2007), 1–6. doi:10.1109/ICARCV.2006.345351.

22. Jansen, T. & Wiegand, R. P. *Exploring the Explorative Advantage of the Cooperative Coevolutionary (1+1) EA* in *Genetic and Evolutionary Computation GECCO 2003* Genetic and Evolutionary Computation Conference (Springer, Berlin, Heidelberg, July 12, 2003), 310–321. ISBN: 978-3-540-40602-0. doi:10.1007/3-540-45105-6_37. https://link.springer.com/chapter/10.1007/3-540-45105-6_37 (2018).

23. Lazaridou, A., Peysakhovich, A. & Baroni, M. Multi-Agent Cooperation and the Emergence of (Natural) Language. *arXiv:1612.07182 [cs].* arXiv: 1612.07182. http://arxiv.org/abs/1612.07182 (2018) (Dec. 21, 2016).

24. Lazaridou, A., Hermann, K. M., Tuyls, K. & Clark, S. Emergence of Linguistic Communication from Referential Games with Symbolic and Pixel Input. https://openreview.net/forum?id=HJGv1Z-AW (2018) (Feb. 15, 2018).

25. Havrylov, S. & Titov, I. Emergence of Language with Multi-agent Games: Learning to Communicate with Sequences of Symbols. *arXiv:1705.11192 [cs].* arXiv: 1705.11192. http://arxiv.org/abs/1705.11192 (2018) (May 31, 2017).

26. Mordatch, I. & Abbeel, P. Emergence of Grounded Compositional Language in Multi-Agent Populations. *arXiv:1703.04908 [cs].* arXiv: 1703.04908. http://arxiv.org/abs/1703.04908 (2018) (Mar. 14, 2017).

27. Lowe, R. *et al.* Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *Neural Information Processing Systems (NIPS)* (2017).

28. Kasai, T., Tenmoto, H. & Kamiya, A. *Learning of communication codes in multi-agent reinforcement learning problem* in *2008 IEEE Conference on Soft Computing in Industrial Applications* 2008 IEEE Conference on Soft Computing in Industrial Applications (June 2008), 1–6. doi:10.1109/SMCIA.2008.5045926.

29. Foerster, J. N., Assael, Y. M., de Freitas, N. & Whiteson, S. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. *arXiv:1605.06676 [cs].* arXiv: 1605.06676. http://arxiv.org/abs/1605.06676 (2018) (May 21, 2016).

30. Kottur, S., Moura, J. M. F., Lee, S. & Batra, D. Natural Language Does Not Emerge 'Naturally' in Multi-Agent Dialog. *arXiv:1706.08502 [cs].* arXiv: 1706.08502. http://arxiv.org/abs/1706.08502 (2018) (June 26, 2017).

31. Clark, H. H. & Brennan, S. E. in *Perspectives on Socially Shared Cognition* (eds Resnick, L., B, L., John, M., Teasley, S. & D.) 13–1991 (American Psychological Association, 1991).

32. Horvitz, E. & Paek, T. Grounding Criterion: Toward a Formal Theory of Grounding (Jan. 1, 2000).

33. Clark, H. H. & Schaefer, E. F. Contributing to Discourse. *Cognitive Science* **13,** 259–294. ISSN: 1551-6709.

34. Koschmann, T. & Lebaron, C. D. *Reconsidering Common Ground: Examining Clark's Contribution Theory* in *In* (Kluwer Academic Publishing, 2003).

35. *Robot Operating System* Collaborative Robotics and Intelligent Systems (CoRIS) Institute. https://robotics.oregonstate.edu/robot-operating-system (2018).

36. *Master - ROS Wiki* http://wiki.ros.org/Master (2018).

37. *PhantomX Pincher Robot Arm Kit* https://www.trossenrobotics.com/p/PhantomX-Pincher-Robot-Arm.aspx (2018).