Adaptability Features in a Nonlinear System-based Swarm Robotic Framework

A Thesis submitted to
Graduate School of Engineering
of the Osaka Prefecture University
in Fulfillment of the Requirement
for the Degree of Master of Engineering

Master Student:
Kevin Denamganai
ID: 2160104105

Supervisor:
Professor Keiji Konishi

Department of Electrical and Information Systems
Osaka Prefecture University
Japan
August 2017

CONTENTS

I INTRODUCTION 6

1 BACKGROUND 7
- 1.1 Swarm Robotics 7
- 1.2 Nonlinear/Complex Systems 7
- 1.3 Deep Reinforcement Learning 8

2 MOTIVATION 9

3 OUTLINE 10

II NONLINEAR/COMPLEX SYSTEMS-BASED SWARM ROBOTIC FRAMEWORK 11

1 INTRODUCTION 12

2 CONTROL LAW 13
- 2.1 Background 13
- 2.2 Formulation 13

3 ARCHITECTURE 16
- 3.1 Reference Frames 16
- 3.2 Distributed Pipeline 17
- 3.3 Simulations 18

4 OBSTACLE AVOIDANCE CONTROL LAW 21
- 4.1 Approach #1: Obstacle Avoidance Control Law with Limit Cycles 21
- 4.2 Approach #2: Obstacle Avoidance Control Law with Radius Evolution 23
- 4.3 Simulations 24
  - 4.3.1 Obstacle Avoidance with one robot 25
  - 4.3.2 Obstacle avoidance with a swarm of robots 26
- 4.4 Discussion 28

5 CONCLUSION 30

III DEEP REINFORCEMENT LEARNING IN NONLINEAR/COMPLEX SYSTEM-BASED SWARM ROBOTIC FRAMEWORK 31

1 INTRODUCTION 32

2 DEEP REINFORCEMENT LEARNING 33
- 2.1 Background 33
- 2.2 Related Works 35
- 2.3 Current Approach 35

3 DEEP REINFORCEMENT LEARNING AND NONLINEAR SYSTEM-BASED COUPLED CONTROLLER 38
- 3.1 Related Works 38
- 3.2 Architecture 38
- 3.3 Reward shaping 39
- 3.4 Simulations 40
  - 3.4.1 Experimental setup 40
  - 3.4.2 Results 41
- 3.5 Discussion 41

4 CONCLUSION 43

Appendices 44
A DISTRIBUTED MODEL 45
A.1 Hardware to solve the DATMO Problem : 45
A.2 Software to solve the DATMO Problem : 45
A.2.1 Classical Computer Vision-based Solution : Planar-based Solution 45
A.2.2 Qualitative Evaluation of the Planar-based Solution 46
A.2.3 Quantitative Static Evaluation of the Planar-based Solution 47
A.2.4 Deep Learning-based Solution : 49
B DDPG-BA2C ALGORITHM: VALIDATION EXAMPLE 52
B.1 Problem Formulation 52
B.2 Effect of Reward Shaping 52
Acknowledgments 56
Bibliography 56
Publications 60

LIST OF FIGURES

Figure 1 Two-wheeled robots in their global reference frame. 13
Figure 2 Vector field of the encircling behavior control law. 14
Figure 3 Architecture of the pipeline of the previous experimental implementation. 15
Figure 4 Distributed architecture of the pipeline in its current implementation. 16
Figure 5 Details on the robot-focused local reference frames $\mathcal{R}_r^i$ and $\mathcal{R}_r^{i+1}$, and on the robot-focused target-centered local reference frames $\mathcal{R}_{target}^i$ and $\mathcal{R}_{target}^{i+1}$. 17
Figure 6 Pipeline implemented in each robot. 17
Figure 7 Current view of the robot model in simulation. 18
Figure 8 Trajectories of 2 simulated two-wheeled robots from initial pose *(black)* to final pose *(grey)*. Parameters : $R = 2$, $\Omega = 2$, $k_\omega = 0.2$, $\epsilon = -1$, $k_v = 0.1$, $a = 1.0$ 19
Figure 9 Trajectories of 3 simulated two-wheeled robots from initial pose *(black)* to final pose *(grey)*. Parameters : $R = 2$, $\Omega = 2$, $k_\omega = 0.2$, $\epsilon = 0.5$, $k_v = -0.1$, $a = -1.0$ 19
Figure 10 Trajectories of 4 simulated two-wheeled robots from initial pose *(black)* to final pose *(grey)*. Parameters : $R = 2$, $\Omega = 2$, $k_\omega = 0.2$, $\epsilon = 0.5$, $k_v = 0.1$, $a = 1.0$ 20
Figure 11 Situation near an obstacle with the obstacle avoidance control law with limit cycles. 21
Figure 12 Vector field of the encircling behavior control law integrating the obstacle avoidance control law. 22
Figure 14 Situation near an obstacle with the approach based on radius evolution. 24
Figure 15 Trajectory of one robot encircling the target with the distributed pipeline. 25
Figure 16 Trajectory of one robot encircling the target while avoiding the obstacle. Coloring accounts for the time step. The control law is the one described in Sec. 4.1. 25
Figure 17 Trajectory of one robot encircling the target while avoiding the obstacle. Coloring accounts for the time step. The control law is the one described in Sec. 4.2. 26
Figure 18 Trajectory of two robots encircling the target with the distributed pipeline. 27
Figure 19 Trajectory of two robots encircling the target while avoiding the obstacle. Coloring accounts for the time step. The control law is the one described in Sec. 4.1. 28
Figure 20 Trajectory of two robots encircling the target while avoiding two obstacles. Coloring accounts for the time step. The control law used is the one described in Sec. 4.2. 29
Figure 21 Reinforcement Learning typical loop of interactions between the environment (world) and the learning agent (brain). 34
Figure 22 Algorithm DDPG-BA2C's pseudocode for the main actor loop. 36

Figure 23 Algorithm DDPG-BA2C's pseudocode for each actor-learner thread. 37
Figure 24 Reinforced NonLinear-Based Neural Controller's architecture. 39
Figure 25 Experimental setup of the proof of concept experiment. 40
Figure 26 Comparison of *RNLBNC*, *NLC*, and *NC* over 400 episodes of 2000 steps each on the target-encircling task with one robot and multiple obstacles. The y-axis shows the total return $R = \sum_t r_t$ for each episode. The higher the total return, the better is the performance. 41
Figure 27 Comparison of *RNLBNC*, *NLC*, and *NC* over 400 episodes of 2000 steps each on the target-encircling task with one robot and multiple obstacles. The y-axis shows the total return $R = \sum_t r_t$ for each episode, scaled in decibel. The higher the total return, the better is the performance. 42
Figure 28 Camera mount (left), fisheye lens (middle) and current state of the robot (right). 46
Figure 29 Example of a recognition of the Raspberry Pi logo relying on the planar-based solution implemented. 46
Figure 36 Test (blue) & train (red) accuracies of a {reluconvx2-16-32+STN+relumaxpoolconvx1-64+128xFC} neural network. 50
Figure 37 Framework of the Cart-pole Problem. 53
Figure 38 Trajectory of the center of mass in the X-Y plane, by episode 600. 53
Figure 39 Trajectory of the center of mass in the X-Y plane, by episode 1000. 54
Figure 40 Trajectory of the center of mass in the X-Y plane, by episode 2000. 54
Figure 41 Trajectory of the center of mass in the X-Y plane, by episode 40, using efficient reward-shaping. 55

Part I

INTRODUCTION

1

BACKGROUND

1.1 SWARM ROBOTICS

Swarm robotics is a subfield of robotics and multi-agent systems. Multi-agent systems can be seen everywhere in nature, from ants, bees and birds to fishes. In robotics, robots are built with goals and tasks that are required to be achieved. Thus, having multiple agents to fulfill some given tasks represents an ideal paradigm for two main reasons. First, the workload can be managed in a parallel fashion, making it astonishingly faster to be fulfilled. Secondly, thanks to the modularity, if one of the agents were to get broken, it can easily be replaced by any other agent of the swarm and the task can be carried on. Given the multi-agent dependency, their is a need to coordinate all the agents in a framework that would be relevant to the given tasks. Following [1][2][3], swarm robotics can exhibit many different framework architectures. The current study only aims at a decentralized/distributed non-communicating leader-free architecture that still relies on a target. This target reliance restricts our study to a limited number of tasks. The field of swarm robotics also exhibits many possible applications and tasks to be performed, but the current study will mainly focus on circular pattern formation around a target as it is the direction that has been assumed by our laboratory's previous works.

Decentralization is a necessary features in order, firstly, to enable us to deploy such a swarm of robots in as many as different situations as possible and, secondly, to unlock swarm intelligence features. We refer to swarm intelligence as the emergence of global behaviors from local interactions [4][5]. Distributed/decentralized architectures require each robot to be equipped with its own on-board sensors in order to monitor its environment. In those varying environments, there is a need to for each robot to account for the other entities of the environment that can be regarded as static or dynamic obstacles [6][7]. The same way we need to guarantee the safety of each robot with regards to extra-swarm obstacles, we need to guarantee intra-swarm safety, thus preventing each robot to collide with its neighbours [8][9][10].

1.2 NONLINEAR/COMPLEX SYSTEMS

Nonlinear and, more broadly, complex systems are systems whose behaviors can be described by autonomous equations and they exhibit a lot of interesting features [11]. From simple analytic formulation, they can describe very complex behaviors, thus their possible concrete applications in real-world problem-solving processes are highly regarded. In this study, we mainly limit ourselves to the the Kuramoto phase-coupled oscillators and synchronization phenomenon they might exhibit [12]. Although a lot of tools to study their stability or bifurcation phenomenon have been devised, there are few concrete applications cases in which they proved themselves to be the best solution. Throughout this thesis, we will look at nonlin-

ear and complex systems through the lens of behavior design. Indeed, similarly to algorithm, nonlinear and complex systems enable us to create behaviors. Thanks to the extensive literature in that field, we have a lot of theoretical tools to study those behaviors and prove their efficiency.

1.3 DEEP REINFORCEMENT LEARNING

Reinforcement learning, as introduced in [13], provides a framework for an agent to learn how to fulfill a given task, through trials and errors. The agent would first explore the many possibilities that its set of actions offers it, to finally exploit the knowledge learned through that exploration in order to effectively fulfill its given task, or achieve its given goal. In that framework, the agent exhibits adaptability features towards the environment that it tries to explore. On its own, this framework remains limited to simple problems, simple tasks because the more complex the environment to explore, the more intractable it becomes for the agent to make sense of how to reach its goal. But, thanks to the recent advances in the combination of deep learning and reinforcement learning, it yields a framework, entitled deep reinforcement learning, with enough representation power in order to handle complex environments and complex tasks.

It started with the Deep Q-Network algorithm [14] that was able to play Atari 2600 games, where the action space is discrete, at a super-human level and it went all the way to classic control problems, where the action space is generally continuous, with the Asynchronous Advantage Actor Critic (A3C) [15] and Deep Deterministic Policy Gradient (DDPG) [16] algorithms. Even though the deep reinforcement learning framework yields impressive results in challenging tasks, it remains to improve on the time efficiency and the data efficiency of the experience sampling during the exploration phase. Compared to human-beings, deep reinforcement learning agent requires an awful amount of time and data processing before achieving similar results. This limitation results in a two-phase process when it comes to the deployment of such solutions: first there is an exploration and training phase, preferably done in a simulated environment, then comes the exploitation phase that sees the agent being deployed and performing on the actual real environment. Because of this exploration phase, made of trials and errors, there are situations in which it is not possible to carry it on the actual real environment, because of its potential high-priced value for instance. Another issue with deep reinforcement learning lies in its dependence to deep learning in the sense that such solution fail to be accounted for, so far. We are very limited when it comes to decipher the processes at play in deep neural networks that lead to some decision makings. Such undesirable feature would prevent us from applying that solution to very critical tasks.

2

MOTIVATION

Our main motivation is to introduce adaptability features into a nonlinear system-based swarm robotic framework. This task will be tackled through the two main approaches as described below.

Firstly, we start by a reformulation of our laboratory's previous works's framework in order to have it more in keeping with a robotic pipeline. Then, we will introduce two obstacle avoidance behaviors designed as nonlinear systems in order to shore up the bridge between nonlinear systems and swarm robotics, following our laboratory's previous works. Thus, we aim at furthering the use cases of nonlinear and complex systems into concrete application cases in robotics.

Secondly, following the acknowledgement of some remaining flaws in our nonlinear system-based swarm robotic framework equipped with obstacle avoidance behaviors, we intent to improve on it. We set out to do so thanks to its coupling with a deep learning-based controller in a deep reinforcement learning framework. This coupling is motivated by the synthesis potential that nonlinear systems and deep reinforcement learning bear. The first one being highly accountable, model-based, can be directly deployed but lacks flexibility, whereas the second one is highly adaptable, model-free and cannot be directly deployed nor accounted for. Thus, our second approach aims at investigating this synthesis and start a bridge between nonlinear/complex systems and deep reinforcement learning.

3

OUTLINE

Part II investigates the design and implementation of nonlinear system-based control laws for swarm robotic with a focus on obstacle avoidance. Chapter II.1 reframes our problems. Chapter II.2 introduces the background works and formalism on which we will rely to. Chapter II.3 reformulates the swarm architecture in a distributed fashion in order to fully use the potential of swarm robotic. It enables the system to be deployed in any environment. We effectively validate that formulation in simulations. Chapter II.4 describes the design and implementation of our nonlinear system-based obstacle avoidance control laws. It introduces safety and adaptability features to the system, with respect to the potential environment's hurdles and between each robot of the swarm. Two control laws are devised and validated in simulations [17] [18]. Chapters II.5 concludes on the results shown in this part.

In part III, after acknowledging both the potential for improvements of the obstacle avoidance control laws devised previously and the synthesis potential between nonlinear system based controllers and deep reinforcement learning controllers, we investigate a novel architecture that we call *Reinforced NonLinear-Based Neural Controller (RNLBNC)*. Chapter III.1 reframes our problems in the coupling of nonlinear system-based controllers and deep reinforcement learning controllers. Chapter III.2 introduces the background of deep reinforcement learning and describes a newly devised deep reinforcement learning algorithm entitled *Deep Deterministic Policy Gradient-Based Asynchronous Actor Critic (DDPG-BA2C)*. Chapter III.3 describes *RNLBNC* and the instance of *DDPG-BA2C* used in a proof of concept experiment conducted on the one robot target-encircling with obstacle avoidance task. Chapters III.4 concludes on our results.

Part II

NONLINEAR/COMPLEX SYSTEMS-BASED SWARM ROBOTIC FRAMEWORK

1

INTRODUCTION

Nonlinear and, more broadly, complex systems are widely studied and show a lot of interesting features that are embodied in concise and analytical formulations [11]. From their ability to make complex behaviors arise from simplicity, their possible concrete applications in real-world problem-solving processes are highly regarded. More specifically, our concern is related to Kuramoto phase coupled-oscillators and their applicability in concrete real world problems. Here lies our main motivation into embedding nonlinear/complex systems into robotic controllers.

We focus on swarm robotics because it can be seen as a complex system. In fact, through swarm intelligence, complex swarm behaviors can emerge from the simplest robot behaviors comparably to the way complex dynamics can emerge from the simplest analytic formulae in the study of nonlinear and complex systems. Moreover, a lot of concrete application cases of swarm robotics, such as surveillance, monitoring, rescue and so on, have been proposed.

Here, we focus on our laboratory's previous works' nonlinear controller and aims at extending its use cases. Firstly, we effectively distribute the previously studied architecture and re-frame it in order to achieve scalability of the system by making it more in touch with a common robotic system: the system is now able to be deployed in any environment. Secondly, we set out to introduce some adaptability features in the said framework by designing obstacles avoidance behaviors that relies on nonlinear systems. In a first extension, we show that a similar control law can be used for static obstacle avoidance and that both behaviors, target-encircling (as defined in our laboratory's previous works) and static obstacle avoidance, can co-exist in a single controller. Finally, following the introduction of a second extension, we compare the two extensions in simulations. Following the introduction of obstacle avoidance behaviors, the system is now able to adapt itself to the surrounding environment in which it is deployed, to a given extent.

2

CONTROL LAW

2.1 BACKGROUND

In our laboratory's previous works, a limit cycle-based control law has been shown efficient in controlling a two-wheeled robot to enclose/encircle a defined target [19]. Secondly, following its experimental validation [20], the control law has been extended and experimentally tested in order to control a swarm of robots via coupled oscillators [21, 22]. Eventually, following a reformulation that makes use of the Kuramoto phase-coupled oscillators, the stability of this control law for swarm robotics has been investigated in the study [23], along with a parameter design process that asserts that stability. So far, only numerical results and a centralized architecture-based experimentation have been undergone.

2.2 FORMULATION

In the following, we lay down the formalism that was used in those previous work and that we will subsequently assume. We consider a swarm of N two-wheeled robots (see Fig. 1): dynamic of robot i is described by

$$\begin{bmatrix} \dot{r}_i \\ r_i\kappa_i \\ \dot{\theta}_i \end{bmatrix} = \begin{bmatrix} \cos\theta_i & 0 \\ \sin\theta_i & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_i \\ \omega_i \end{bmatrix}, \tag{1}$$

where $r_i > 0$ is the radial distance, θ_i is the orientation of the robot as defined in Fig. 1, and κ_i is the angular velocity of robot i with respect to the origin of the global reference frame (centered on the target of the encircling behavior). The same as in previous study [23], the origin of the reference frame is also the position of the target that the swarm aims at encircling.

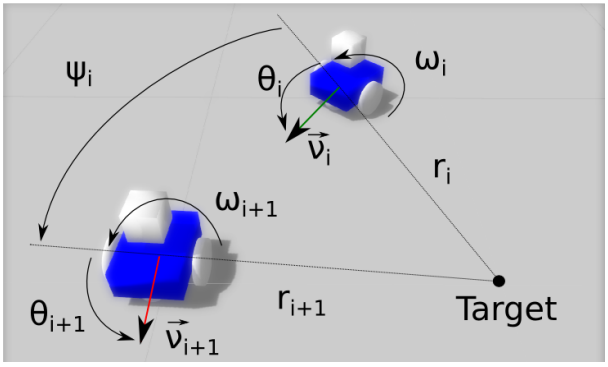

Figure 1.: Two-wheeled robots in their global reference frame.

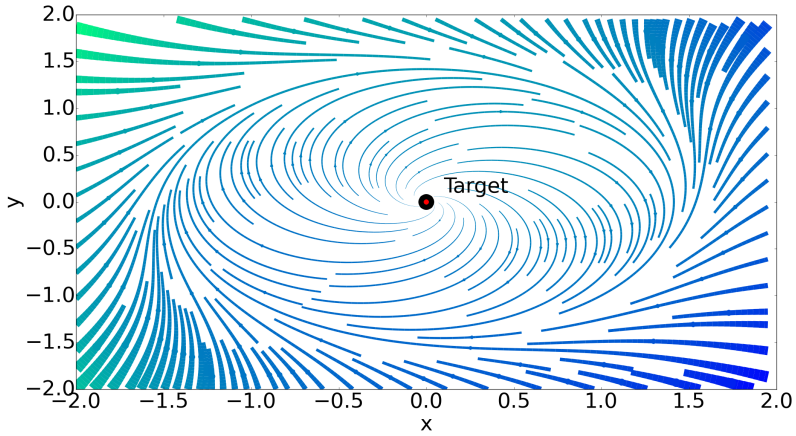

Figure 2.: Vector field of the encircling behavior control law.

In the rest of this paper, we will denote this target-centered reference frame as the global reference frame. v_i and ω_i are the control input computed by our nonlinear control law:

$$v_i = k_v \Big\{ f(r_i, \hat{r}_i) \cos\theta_i + r_i g_{\Omega,\epsilon}(\psi_i) \sin\theta_i \Big\}, \tag{2}$$

$$\omega_i = k_\omega \Big\{ r_i g_{\Omega,\epsilon}(\psi_i) \cos\theta_i - f(r_i, \hat{r}_i) \sin\theta_i \Big\}. \tag{3}$$

This control law relies on two main quantities. Firstly, it depends on the radial velocity, that is given by

$$\dot{r}_i = f(r_i, \hat{r}_i) = a r_i \left(1 - \frac{{r_i}^2}{\hat{r}_i^2} \right), \tag{4}$$

where $\hat{r}_i$ is the desired radial distance at which robot i should encircle the target. a is the constant parameter describing the strength of the distance constraint on the encircling behavior.

Secondly, it relies on the angular velocity with respect to the origin,

$$\kappa_i = g_{\Omega,\epsilon}(\psi_i) = \Omega + \epsilon \sin\psi_i, \tag{5}$$

where Ω is the natural frequency of the associated oscillator, and ϵ is the coupling strength between the coupled oscillators i and $i+1$. The sign of Ω controls the direction of rotation, either clockwise or counter-clockwise, but it is not the only way to change the rotation sense.

Figure 2 shows the vector field induced by control law (2)(3). For more details on the control law and its parameters, refer to our previous study [22]. It is important to mention that the computation of the control law requires the estimation of r_i, θ_i, and ψ_i, which are defined with respect to the global reference frame. We will detail their estimation procedure in the following sections. With regards to the implementation details of the system, in previous experimentation [20], it was relying on a centralized pipeline, with a camera mounted on the top of the experiment area as the global unique sensor, as presented in Fig. 3. The main issues with this architecture are that, firstly, it prevents the swarm to be deployed in any other environment and, secondly, the computational cost grows linearly as the number of robot in the swarm increases. That is to say that the previous architecture was not scalable nor lightweight, whereas those are both desired features when dealing with a swarm of robots.

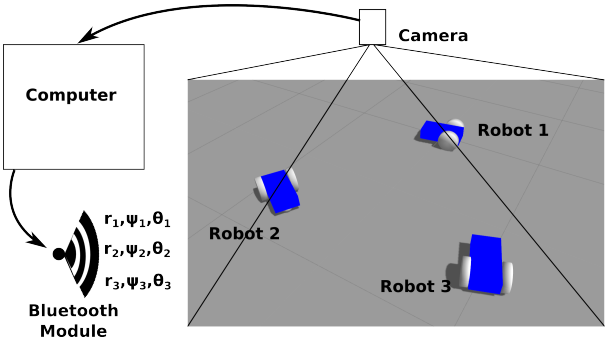

Figure 3.: Architecture of the pipeline of the previous experimental implementation.

3

ARCHITECTURE

3.1 REFERENCE FRAMES

In the current approach, we try to alleviate on the scalability and lightweight issues by decentralizing the pipeline as presented in Fig. 4. As explained previously, robot i needs to estimate the three main quantities defined in the global reference frame, r_i, θ_i, and ψ_i. The issue here is that we no longer have access to that global reference frame since we do not rely on the unique global sensor. In order to estimate them in a decentralized way, each robot needs to be equipped with a camera-typed sensor that enables it to observe its surroundings and thus track the target and its neighbour robots so as to decide with which one to couple at the oscillator level [23]. We now rely on multiple robot-focused local reference frames as described in Fig. 5.

Firstly, let us define the robot-focused robot-centered reference frames $\mathcal{R}_r^i=(x_r^i, y_r^i, z_r^i)$: this reference frame depends on the dynamical frame of robot i. z_r^i points upward while x_r^i is driven by the forward direction of the robot i: $x_r^i = v_i/\|v_i\|$. Thus, y_r^i is defined in a right-handed manner. Secondly, whenever the target is observable by robot i, we can define a robot-focused target-centered reference frame $\mathcal{R}_{target}^i = (x_{target}^i, y_{target}^i, z_{target}^i)$. In this last reference frame, we will re-frame every estimated quantities, r_i, θ_i, and ψ_i, in order to compute the control law (2)(3). Indeed, even though $\mathcal{R}_{target}^i$ rotates with robot i, this reference frame is similar to the previously used global reference frame. By similar, we mean that the required quantities, r_i, θ_i, and ψ_i, can be defined the same way. We can note that, in $\mathcal{R}_{target}^i$, robot i is motionless.

From $\mathcal{R}_r^i$, $\mathcal{R}_{target}^i$ is created from a rotation of $-\theta_i = -(\pi - \theta_{target}^i)$, where θ_{target}^i is the angular offset between the forward direction of robot i and the direction from robot i to the target (see Fig. 5), followed by a translation of $\begin{bmatrix}-r_i & 0 & 0\end{bmatrix}^T$. Thus, for any point $p_{\mathcal{R}_r^i}$ in $\mathcal{R}_r^i$ in

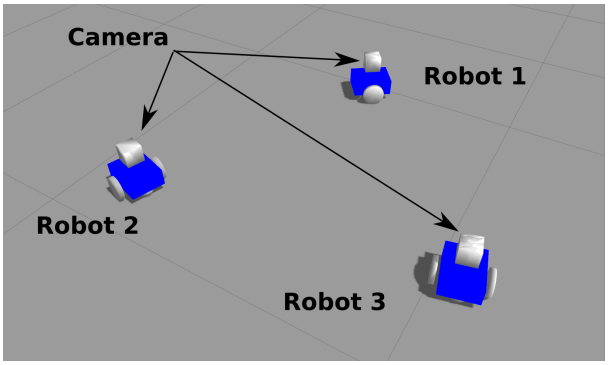

Figure 4.: Distributed architecture of the pipeline in its current implementation.

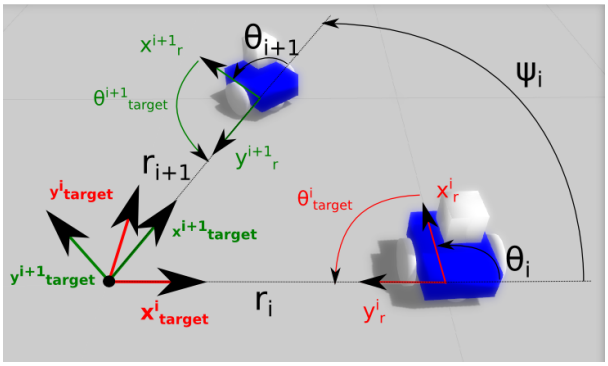

Figure 5.: Details on the robot-focused local reference frames $\mathcal{R}_r^i$ and $\mathcal{R}_r^{i+1}$, and on the robot-focused target-centered local reference frames $\mathcal{R}_{target}^i$ and $\mathcal{R}_{target}^{i+1}$.

its homogeneous form, its transformation into the reference frame $\mathcal{R}_{target}^i$ is achievable as follows:

$$p_{\mathcal{R}_{target}^i} = \begin{bmatrix} R(\theta_{target}^i - \pi) & \begin{matrix} -r_i \\ 0 \end{matrix} \\ \begin{matrix} 0 & 0 \end{matrix} & 1 \end{bmatrix} p_{\mathcal{R}_r^i}, \tag{6}$$

$$R : \phi \mapsto \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix}.$$

This transformation is used to re-frame every surrounding objects into the target-centered reference frame where the desired quantities for the computation of the control law can be measured. That way, the desired quantities, r_i, θ_i, and ψ_i, are measured while being in keeping with their previous definitions that was relying on the global reference frame.

3.2 DISTRIBUTED PIPELINE

Since each robot implements its own sensing devices, the whole pipeline is distributed into each robot, as shown in Fig. 6. This pipeline is built around the robotic paradigm **SENSE-THINK-ACT**.

As explained earlier, each robot is equipped with its own sensors, to wit a fisheye-lensed camera, which feeds a video stream to the **THINK**-related module.

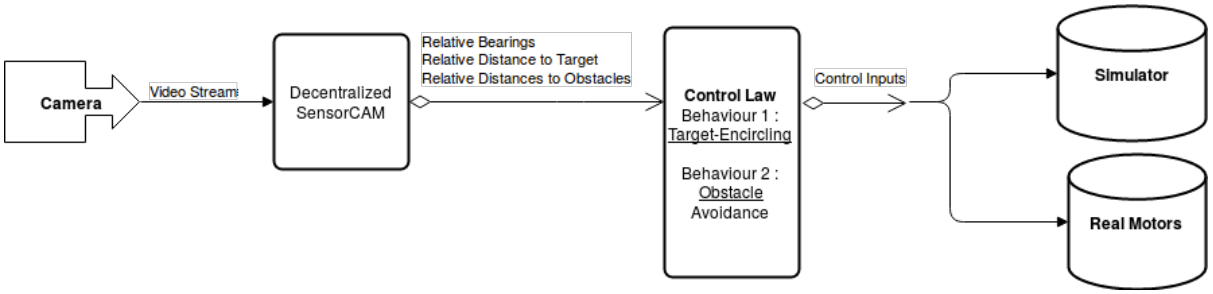

Figure 6.: Pipeline implemented in each robot.

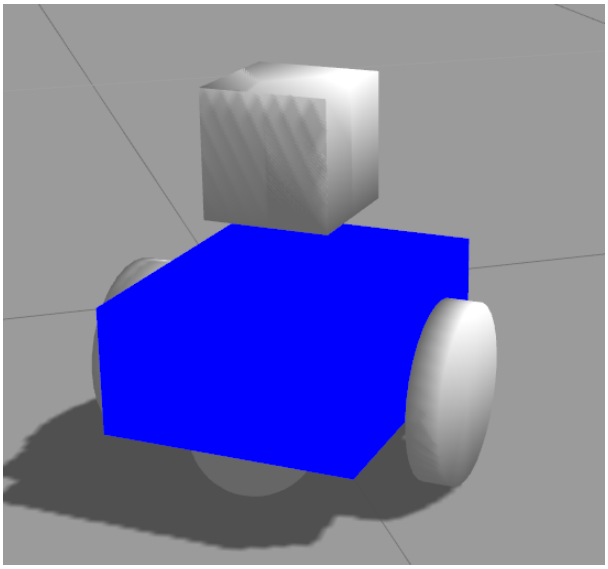

Figure 7.: Current view of the robot model in simulation.

The *Decentralized SensorCAM* module is the core of the **THINK** part of the pipeline. It is a very critical module whose goal is to extract from the video stream the positions of the neighbour robots that are visible and the position of the target in the local reference frame $\mathcal{R}_r^i$, robot-focused and robot-centered. Subsequently, with the introduction of the obstacle avoidance behaviors, this module will also be tasked to retrieve and monitor the obstacle's positions.

The **ACT**-related module is tasked to compute the control laws from the necessary quantities that were provided by the **THINK**-related module. For the moment, each behavior of the swarm of robot is implemented as its own small stand-alone module within it. We will dwell on those small behavior-dedicated modules in the next chapters.

3.3 SIMULATIONS

The previously explained distributed architecture is further tested in a robotic physic-based simulator entitled *Gazebo*[1] and powered by *ROS (Robotic Operating System)*[2]. Robots are modeled as shown in Fig. 7 with, on the top, an omnidirectional camera. We leave it to later further inquiries to test with a fisheye-lensed camera mounted on a rotating mount instead. The real implementation has been undertaken and is described in *Appendix A*.

Figures 8, 9 and *10* demonstrate trajectories undertaken by robots within swarms of, respectively, two, three and four robots. We can see, mainly in Figs. 8 and 9, the effect of the synchronization phenomena as, at the end of the simulation, each robot is equally spaced out from its neighbors on the circular trajectory. As shown in Fig. 10, as the number of robot within the swarm increases, the synchronization phenomena takes more time to be completed, and thus, at the end of this experiment, the robots are not entirely equally spaced out from each other on the trajectory.

1 http://gazebosim.org/
2 http://www.ros.org

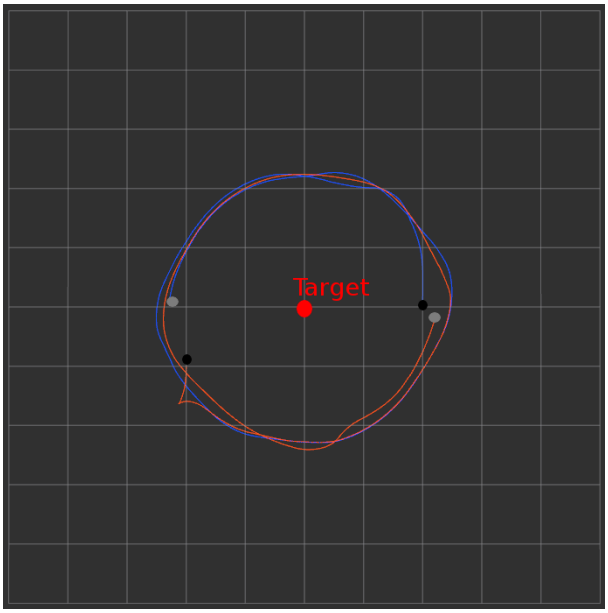

Figure 8.: Trajectories of 2 simulated two-wheeled robots from initial pose *(black)* to final pose *(grey)*. Parameters : $R = 2$, $\Omega = 2$, $k_\omega = 0.2$, $\epsilon = -1$, $k_v = 0.1$, $a = 1.0$

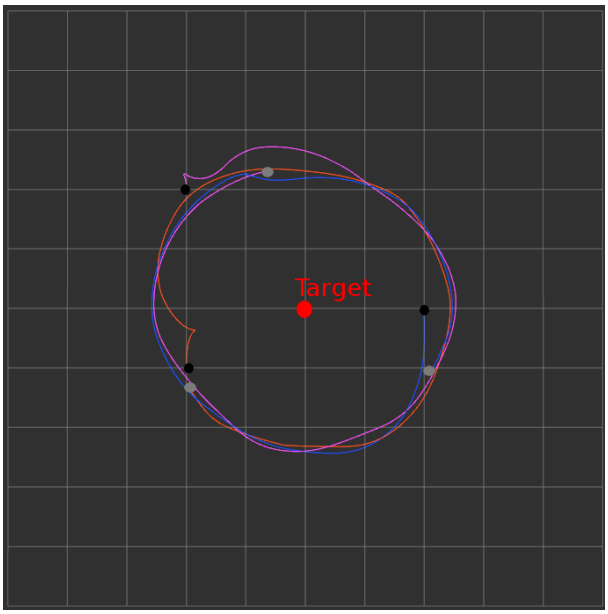

Figure 9.: Trajectories of 3 simulated two-wheeled robots from initial pose *(black)* to final pose *(grey)*. Parameters : $R = 2$, $\Omega = 2$, $k_\omega = 0.2$, $\epsilon = 0.5$, $k_v = -0.1$, $a = -1.0$

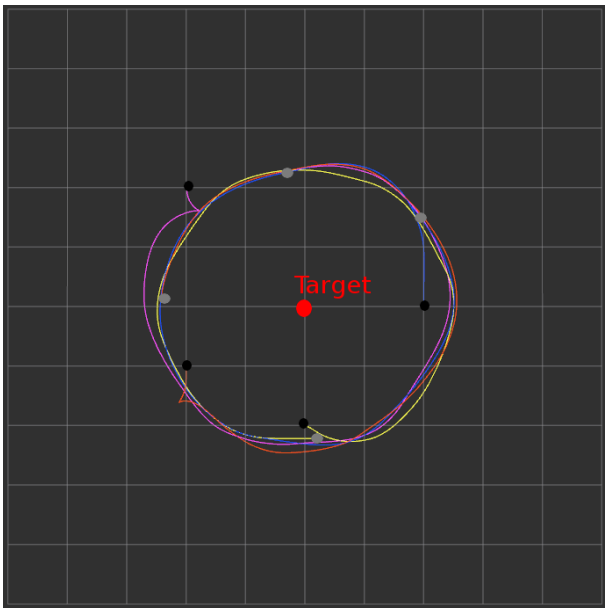

Figure 10.: Trajectories of 4 simulated two-wheeled robots from initial pose *(black)* to final pose *(grey)*. Parameters : $R = 2$, $\Omega = 2$, $k_\omega = 0.2$, $\epsilon = 0.5$, $k_\nu = 0.1$, $a = 1.0$

4

OBSTACLE AVOIDANCE CONTROL LAW

Having successfully distributed the swarm architecture, we now set out to tackle the design of obstacle avoidance behaviors based on nonlinear systems. In this chapter, we detail the two possible extensions/approaches that have been devised before comparing them in simulations.

4.1 APPROACH #1: OBSTACLE AVOIDANCE CONTROL LAW WITH LIMIT CYCLES

With regards to the obstacle avoidance behavior, in this first attempt, we basically implement our previous control law with the observed obstacle as the origin of the reference frame. Let us define both r^i_{obs} and θ^i_{obs} as, respectively, the distance between robot i and its nearest observable obstacle, and the angle between the forward direction of robot i and the direction from robot i to the same obstacle (see Fig. 11). $\hat{r}^i_{obs}$ is the desired distance at which the robot i circles around the obstacle. We assume that r^i_{obs} and θ^i_{obs} can be estimated thanks to the camera-typed sensor that is mounted on robot i.

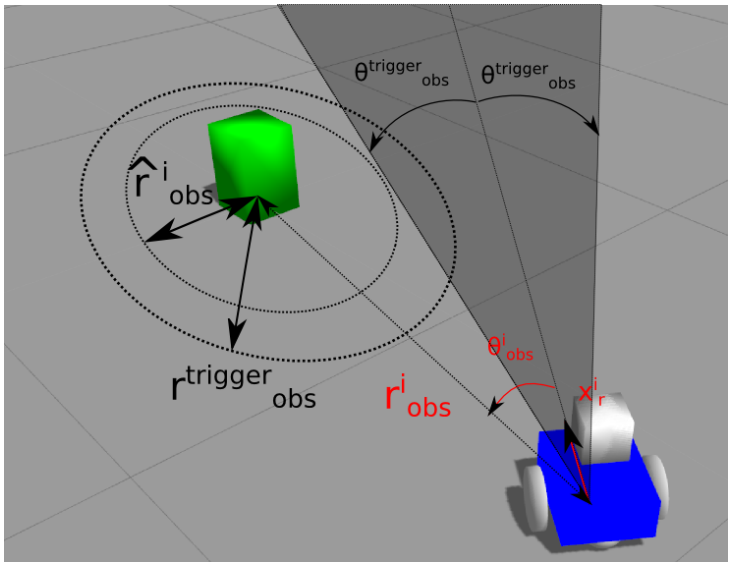

Figure 11.: Situation near an obstacle with the obstacle avoidance control law with limit cycles.

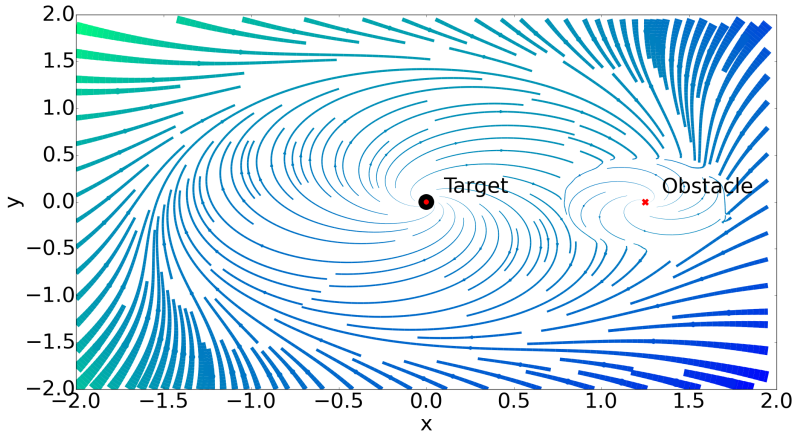

Figure 12.: Vector field of the encircling behavior control law integrating the obstacle avoidance control law.

We focus on the criteria for toggling the obstacle avoidance behavior. This behavior is triggered if both of

$$r^i_{obs} < r^{trigger}_{obs}, \tag{7}$$

$$|\theta^i_{obs}| < \theta^{trigger}_{obs}, \tag{8}$$

hold, with the threshold distance $r^{trigger}_{obs} = 3\hat{r}^i_{obs}/4$ and the angular threshold $\theta^{trigger}_{obs} = 3\pi/8$, for instance. During this obstacle avoidance behavior, the applied control law is as follows:

$$v^{obs}_i = k_v \left\{ f(r^i_{obs}, \hat{r}^i_{obs}) \cos\theta^{obs}_i + r_i g_{\Omega_{obs},0}(\psi_i) \sin\theta^{obs}_i \right\}, \tag{9}$$

$$\omega^{obs}_i = k_\omega \left\{ r^i_{obs} g_{\Omega_{obs},0}(\psi_i) \cos\theta^{obs}_i - f(r^i_{obs}, \hat{r}^i_{obs}) \sin\theta^{obs}_i \right\}, \tag{10}$$

where $\theta^{obs}_i = \pi - \theta^i_{obs}$. Hence, Ω_{obs} is the desired velocity of the rotation around the obstacle.

When the trajectory of robot i has converged on its limit-cycle, we have $|\theta^i_{obs}| = \pi/2$. Once it is asserted that robot i is currently converging on its limit-cycle which enables it to circle around the obstacle, it remains to seek the proper moment at which robot i should evade that limit-cycle. That is to say the proper moment for robot i to re-attached its trajectory on the main limit-cycle. This moment is characterized by a target-encircling behavior-related angular velocity ω_i of lesser magnitude. We formulate it as follows:

$$|\omega_i| \leq \alpha |a k_\omega \hat{r}_i|, \tag{11}$$

where $\alpha > 0$ is a normalizing constant. Indeed, when the robot is approaching the desired trajectory, we have $\theta_i \to \pi/2$ and $r_i \to \hat{r}_i$. Thus, it ensures that $f(r_i, \hat{r}_i) \to a\hat{r}_i \mathcal{O}(r_i - \hat{r}_i)$ and, obviously, $\omega_i \to -k_\omega a \hat{r}_i \mathcal{O}(r_i - \hat{r}_i)$, where $\mathcal{O}$ refers to the "little o" notation. In that sense, at some point $|\omega_i|$ will practically satisfy inequality (11). We show on Fig. 12 the vector field under which the swarm's robots will behave given the presence of an obstacle on the robots' limit cycle trajectory. Intuitively, this evasion criteria is tantamount to the moment where the vector fields of both the dynamic related to the target encircling behavior and the dynamic

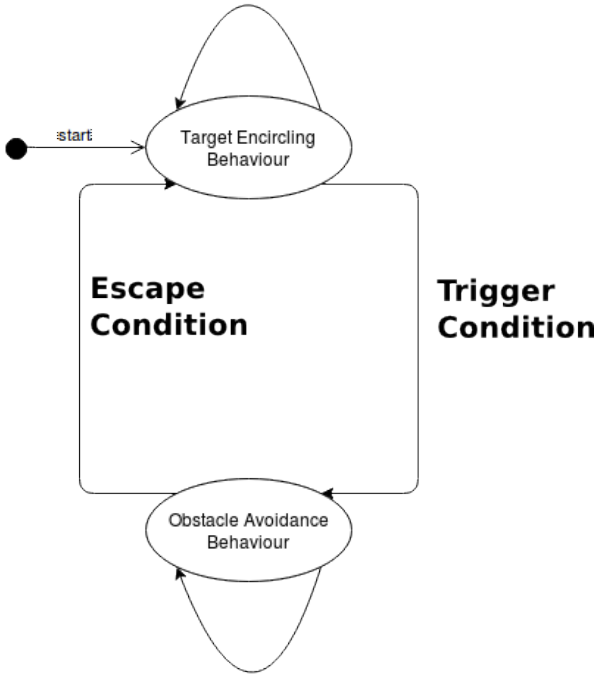

Figure 13.: State machine that manages the transitions between each behavior.

related to the obstacle avoidance behavior are aligned.

Finally, it is necessary to introduce a state machine scheme in order to manage those two behaviors and their transition conditions, as shown in Fig. 13.

4.2 APPROACH #2: OBSTACLE AVOIDANCE CONTROL LAW WITH RADIUS EVOLUTION

We provide a second obstacle avoidance control law which is less crafty than the previous one since it no longer requires a state-machine approach. It relies on the previously defined control law for the target-encircling behavior [23] with a simple extension: the radius $\hat{r}_i$ is no longer constant. It is updated along time t via the following equation:

$$\dot{\hat{r}}_i(t) = \{1 - \cos([\theta^i_{obs}(t)])\} h_1\Big(\hat{r}_i(t)\Big) - \cos([\theta^i_{obs}(t)]) h_2\Big(r^i_{obs}(t), [\theta^i_{obs}(t)]\Big), \tag{12}$$

where h_1 and h_2 are defined as follows:

$$h_1\Big(\hat{r}_i(t)\Big) = k_{\hat{r}}\{\hat{\hat{r}}_i - \hat{r}_i(t)\}, \tag{13}$$

$$h_2\Big(r^i_{obs}(t), [\theta^i_{obs}(t)]\Big) = \frac{sign([\theta^i_{obs}])}{\log\{1 + k_{\hat{r}}|r^{trigger}_{obs} - r^i_{obs}(t)|\}}, \tag{14}$$

where

$$sign \,:\, x \mapsto \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}, \tag{15}$$

$$[\,] \,:\, \theta \mapsto \begin{cases} \frac{\pi}{2} & \text{if } |\theta| \geq \frac{\pi}{2} \\ \theta & \text{otherwise} \end{cases}. \tag{16}$$

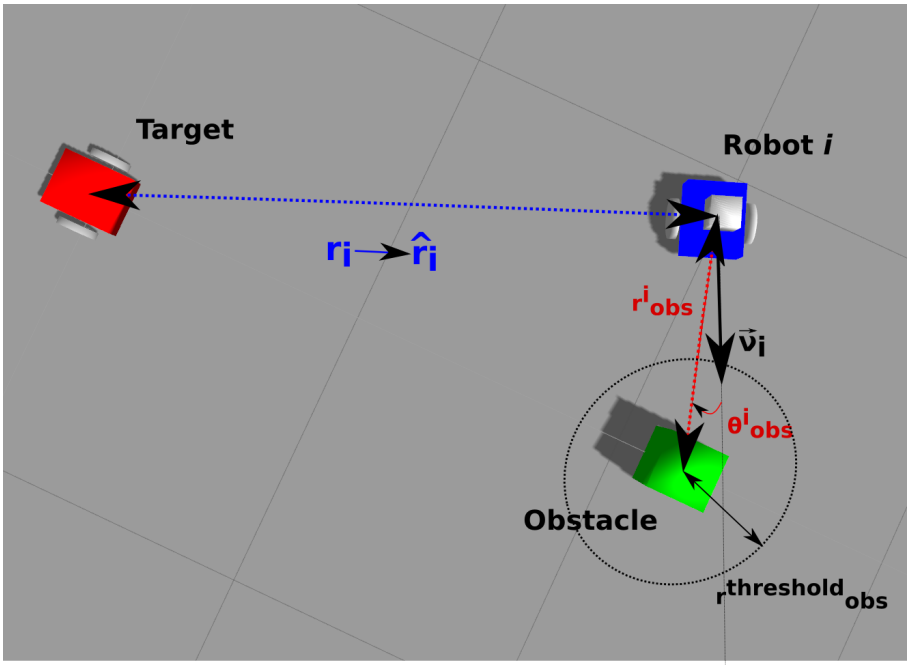

Figure 14.: Situation near an obstacle with the approach based on radius evolution.

$\hat{\hat{r}}_i$ is now the constant desired radius parameter for which robot i should encircle the target. h_1 aims at the convergence of $\hat{r}_i$ towards $\hat{\hat{r}}_i$ over time, whereas h_2 controls the reduction and extension of the radius value in order prevent the trajectory of robot i from going through the location of the nearest observable obstacle. $k_{\hat{r}}$ is a parameter that controls the strength applied to the convergence constraint toward the desired radius $\hat{\hat{r}}_i$.

Figure 14 shows the situation near an obstacle when this approach is applied. We can think of that second approach in a hierarchical way as being made of a stacking of nonlinear controllers:

- Level 1: the user fixes the parameter $\hat{\hat{r}}_i$ that stands for the desired radius of the target-encircling behavior.
- Level 0: *Eq. 12* controls the radius parameter $\hat{r}_i$'s evolution in order to avoid obstacles. Asymptotically, we have
$$\hat{r}_i \to \hat{\hat{r}}_i.$$
- Level -1: the nonlinear controller for the target-encircling behavior makes use the radius parameter $\hat{r}_i$. And like before, asymptotically, we have
$$r_i \to \hat{r}_i.$$

4.3 SIMULATIONS

This section will now details simulations that have been set out to compare the two previously defind approaches. All simulations have been carried on using *ROS*[1] in the robotic physic-based simulator *Gazebo*[2] with simulated robots which comprise PI-controllers[3]. Each

1 `http://www.ros.org`
2 `http://gazebosim.org/`
3 Source code is made available here: `https://github.com/Near32/OPU_Sim/`

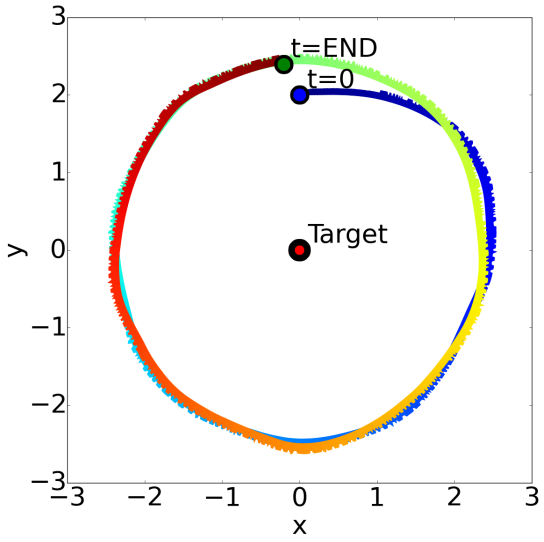

Figure 15.: Trajectory of one robot encircling the target with the distributed pipeline.

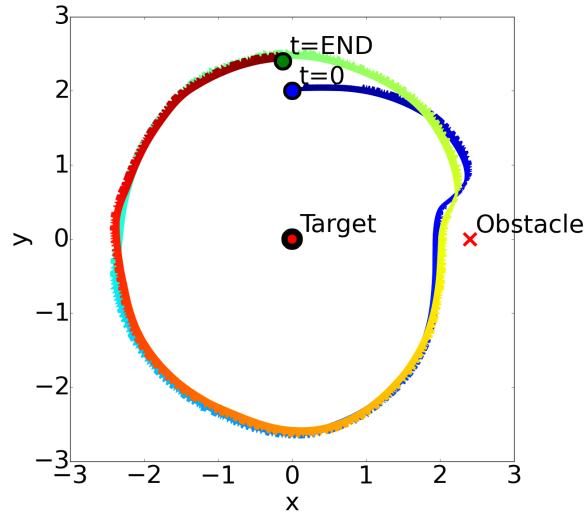

Figure 16.: Trajectory of one robot encircling the target while avoiding the obstacle. Coloring accounts for the time step. The control law is the one described in Sec. 4.1.

robot employs an omnidirectional camera as unique sensor.

4.3.1 *Obstacle Avoidance with one robot*

We demonstrate the ability of the control law to exhibit the encircling behavior while avoiding obstacles. Figure 15 shows the trajectory of one robot encircling the target as a baseline. Figure 16 shows the trajectory of one robot encircling the target while an obstacle that is meticulously set on its path. Comparing Fig. 15 to Fig. 16 gives a sense of the deformation induced by the presence of a static obstacle under the use of the control law explained in Sec. 4.1. For this experiment, the following values were used: $\alpha = 0.1$, $\Omega = 2.0$, $k_\omega = 0.2$, $k_v = -0.1$, $a = -1.0$, $\hat{r}_i = 2.0$, and $\hat{r}^i_{obs} = 0.6$.

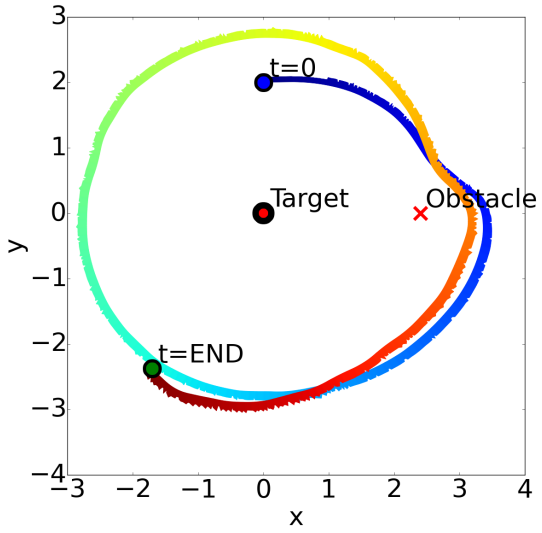

Figure 17.: Trajectory of one robot encircling the target while avoiding the obstacle. Coloring accounts for the time step. The control law is the one described in Sec. 4.2.

The deformation induced by the static obstacle on the trajectory of the robot is, as expected, similar to the vector field shown in Fig. 12. Looking more closely on the part of the trajectory where the robot evades the limit cycle of the obstacle avoidance behavior to re-attach on the main limit cycle, we can see that the convergence of the distance of the robot from the target towards the desired radial distance $\hat{r}_i$ takes almost a quarter of the circle. In the future, it could be interesting to investigate the time it takes for a robot to recover from the obstacle avoidance behavior.

Figure 17 shows the trajectory of one robot encircling the target while an obstacle that is meticulously set on its path. Comparing Fig. 15 to Fig. 17 gives a sense of the deformation induced by the presence of a static obstacle under the use of the control law explained in Sec. 4.2. For this experiment, the following values were used: $k_{\hat{r}} = 10$, k_{obs}, $\Omega = 2.0$, $k_\omega = 0.2$, $k_v = -0.1$, $a = -1.5$, $\hat{\hat{r}}_i = 2.0$, and $\hat{r}^i_{obs} = 0.6$.

4.3.2 *Obstacle avoidance with a swarm of robots*

We demonstrate that adding an obstacle avoidance behavior does not impair the swarm of robots to converge on the desired circular formation pattern with the maximization of each agent utility. Indeed, in that situation, robots are not only required to encircle the target while avoiding obstacles, they are also required to be equally spaced out on the circular trajectory. This adds a degree of difficulty compared to the previous experimental situation.

Figure 18 shows the trajectories of two robots encircling the target as a baseline while Fig. 19 shows the trajectories of two robots encircling the target and avoiding obstacles. Comparing those two figures gives a sense of the deformation induced by the presence of a static obstacle under the use of the control law explained in Sec. 4.1. For this experiment, the following values were used: $\alpha = 0.1$, $\Omega = 2.0$, $k_\omega = 0.2$, $\epsilon = 1.0$, $k_v = 0.1$, $a = 1.0$, $\hat{r}_i = 2.0$, and $\hat{r}^i_{obs} = 0.6$.

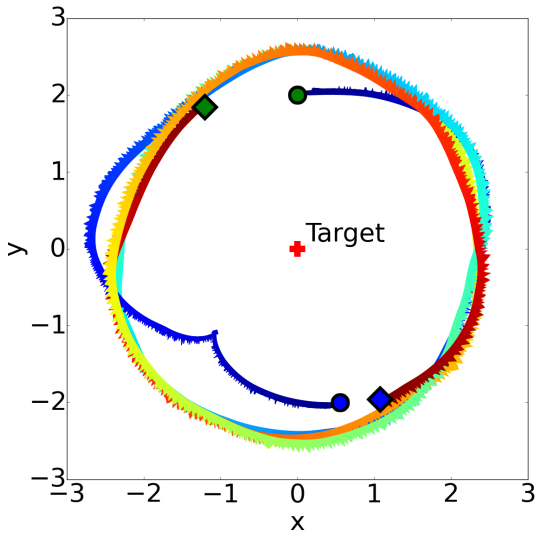

Figure 18.: Trajectory of two robots encircling the target with the distributed pipeline.

By focusing on the color at each time step of each trajectory, we can see that the formation is indeed converging to a state where each robot is equally spaced from the other around the circle. Here, since there are two robots, the phase difference is converging toward π. By focusing more closely on Fig. 19, we can see that, even in presence of an obstacle, the previously explained formation pattern is achieved: robots are converging towards a state where they are equally space from each other around the circle.

In Fig. 20, two obstacles are set on the robots' trajectories. The obstacle on the right is positioned at a distance from the target that is slightly under the current value of $\hat{\hat{r}}_i = 2.0$, whereas the obstacle on the left is positioned at a slightly greater distance than $\hat{\hat{r}}_i = 2.0$. It is important to note that those distances are to be understood within the reference of the estimation made by the robots: those estimation being not perfect yet, when the robots estimate their distance from the target to be at 2.0, they are in reality more around 2.45 as seen in every presented figures. Comparing Fig. 18 to Fig. 20 gives a sense of the deformation induced by the presence of a static obstacle under the use of the control law explained in Sec. 4.2. For this experiment, the following values were used: $k_{\hat{r}} = 10$, k_{obs}, $\Omega = 2.0$, $k_\omega = 0.2$, $k_v = -0.1$, $a = -1.5$, $\hat{\hat{r}}_i = 2.0$, and $\hat{r}^i_{obs} = 0.6$. An interesting feature of that extension is that it inherently integrates the possibility to go over the obstacle or under it, depending on the initial angle under which the obstacle is seen from robot i's camera sensor.

Unfortunately, the extension explained in Sec. 4.2 is not successful enough for the moment, as Fig. 20 shows far from optimal trajectories near the obstacles at the end of the simulation. With regards to the extension explained in Sec. 4.1, the robots avoid obstacles by rotating counter-clockwise around it, as the vector field in Fig. 12 shows. In the future, we shall tackle the issue with regards to the rotation sense around the obstacle so that the trajectory would be the shortest possible, for instance.

Finally, further inquiries will be made in the presence of multiple static and dynamical obstacles, once the highlighted issues will have been solved.

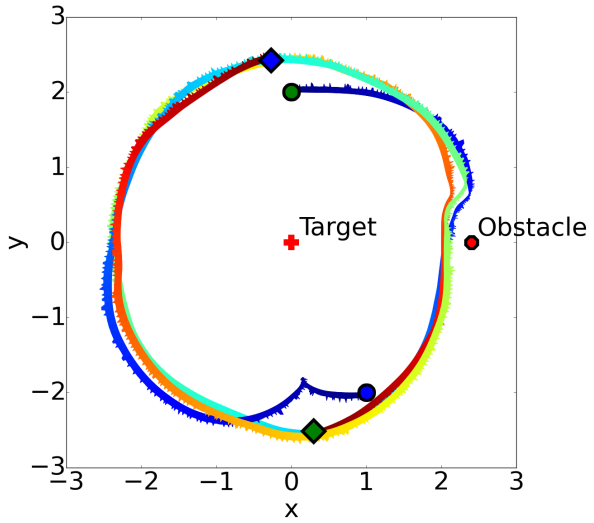

Figure 19.: Trajectory of two robots encircling the target while avoiding the obstacle. Coloring accounts for the time step. The control law is the one described in Sec. 4.1.

4.4 DISCUSSION

The first approach has proposed an obstacle avoidance control law based on a limit cycle system. The efficiency of the control law has been validated in a robotic physic-based simulator entitled *Gazebo*. Although it has been proven efficient, we have had to employ a state machine-based approach in order to manage the switch between each behavior. Because of that state machine, it is no longer possible to apply control theory tools to analyze the control law's stability, for instance. We end up with an hybrid system that somehow combines the best of classical robotics, one might say, with the best of nonlinear systems-based controllers, in practice.

With regards to the second approach, the efficiency have been validated but there remains some situation in which the controller fails to drive safely the robot. Nevertheless, we consider the second approach to be of a greater interest than the first one. Indeed, one of the main advantages of that second obstacle avoidance control law is that it integrates both possible avoidance trajectories (either over or under the obstacle) that is intrinsically controlled by the angle θ^i_{obs} under which the obstacle is observed initially by robot i's camera sensor. Secondly, we end up with a framework that relies solely on nonlinear systems-based controllers, thus, making it possible to analyze the system's stability with some well-known control theory's tools in a subsequent work.

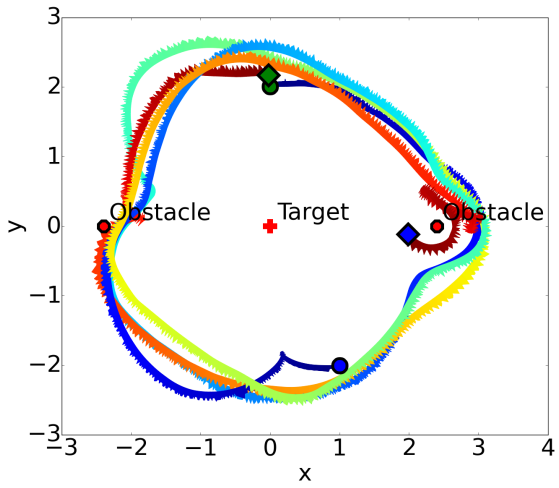

Figure 20.: Trajectory of two robots encircling the target while avoiding two obstacles. Coloring accounts for the time step. The control law used is the one described in Sec. 4.2.

5

CONCLUSION

We have reformulated our laboratory's previous works's framework in order to have it more in touch with a robotic pipeline and a distributed swarm architecture. It ensued the introduction of two obstacle avoidance behaviors designed as nonlinear system-based controllers in order to shore up the bridge between nonlinear systems and swarm robotics. Following the comparison of those two approaches, we acknowledge their partial efficiency and potential for improvements that we will pursue in the next part.

Part III

DEEP REINFORCEMENT LEARNING IN NONLINEAR/COMPLEX SYSTEM-BASED SWARM ROBOTIC FRAMEWORK

1

INTRODUCTION

As stated earlier, we can acknowledge a gap between biologically-inspired approaches and control theory-based approaches, to which the nonlinear/complex systems-based approach belongs, when one tackles robotic issues. On the contrary to that, our motive here is to find a fair trade-off between those two highly studied approaches.

Using deep reinforcement learning in a framework enables it to show great flexibility and adaptability with regards to the goals of the systems and the hurdles that it may face down its quest. Yet, on the down-side, deep reinforcement learning agents requires a lot of trials and errors before being operational, as the current state-of-the-art is not able to exhibit efficient sampling of its experiences yet. Moreover, being based on deep neural networks, it fails to be accountable. We are extremely limited in the task of explaining its decision-making processes.

On the other side, nonlinear/complex systems-based controllers are extremely accountable for a lot of tools aiming to investigate their behaviors and stability have been developed. But their adaptability potential is scarily limited. As shown in the previous parts, introducing nonlinear systems-based obstacle avoidance behaviors has shown itself to be quite a challenging task already.

Given the synthesis potential of deep reinforcement learning and nonlinear systems-based controller, as presented above, we set out to investigate multiple coupling architectures in the swarm robotics framework introduce previously. Given state-of-the-art model-free algorithms of deep reinforcement learning and model-based (obviously) nonlinear systems-based controller, we expect to observe a better experience sampling efficiency of the learning agents as well as an accelerated learning phase time before reaching an optimal policy. Finally, such a coupling could lead us to a deep reinforcement learning-based solution that would no longer require any learning phase separated from the exploitation phase given the seemingly out-of-the-box near-optimality of the nonlinear system-based controller it consists of.

2

DEEP REINFORCEMENT LEARNING

2.1 BACKGROUND

Firstly, let us detail the challenges encountered when one try to apply deep neural network to the Reinforcement Learning (RL) realm, to wit :

1. The scalar reward in RL algorithm is, in most cases, frequently sparse, noisy and delayed. It is a far more difficult situation than to train a neural network in a supervised way as usual.
2. During reinforcement learning process, data are typically sequences of highly correlated states. Yet, when training neural networks, we need to assert that batches of element on which the network is learning are uncorrelated, otherwise our gradient descent-based algorithms will not converge to global minima.
3. In RL, due to the exploration of the state and action spaces, the data distribution varies. This can be very daunting for neural networks to learn from since their weights update is a rather slow process, compared to how quickly the state and action distributions might change.

Strong of those issues that will be dealt with using multiple related works, let us define the reinforcement learning formalism (see Fig. 21) more precisely as follows :

- An agent is interacting with an environment $\mathcal{E}$.
- At each time step t, the agent observes a state/observation $s_t / x_t \in \mathbb{R}^d$ from the environment $\mathcal{E}$.
- At each time step t, the agent sends an action $a_t \in \mathcal{A}$ to the environment.
- Finally, at each time step t, the agent receives a reward r_t representing the score in the current game/emulator/environment.

In order to apply reinforcement learning methods for Markov Decision Process (MDP), we choose to define the state of the learning agent $s_t = x_1, a_1, \ldots, a_{t-1}, x_t$. The benefits of this formulation are twofolds:

- In the situation in which the screen x_t would be the same as the screen x_{t-k}, for some $k \in [1, t]$, the state s_t can be regarded as different state, whereas it would not have been the case if $s_t = x_t$ were to be used as the state definition.
- The learning agent needs to be able to retrieve velocity-related data from the state s_t, for instance the speed of the ball in a game of *Pong*. The current state definition makes it possible since it encompasses x_t and x_{t-1}.

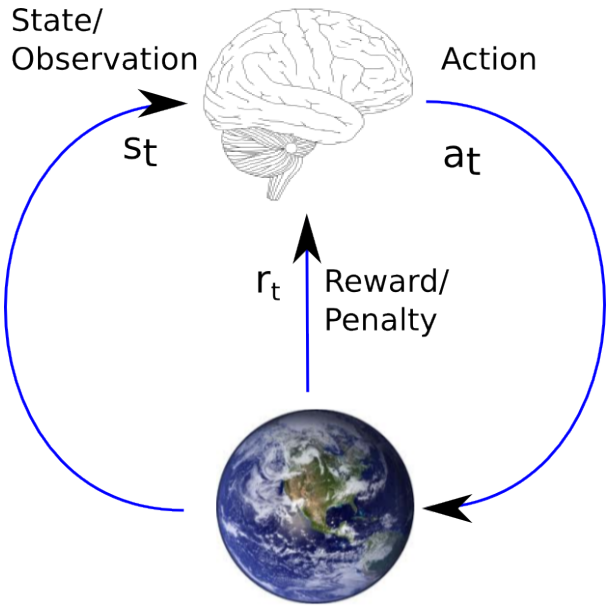

Figure 21.: Reinforcement Learning typical loop of interactions between the environment (world) and the learning agent (brain).

From that, let us define the future discounted return at time t:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}, \tag{17}$$

where γ is the discount factor and T is the time-step at which the game/environment/simulation reaches an end. That end can be either the completion of a task or the fact that time as ran out. The main goal is to find a policy π that maximizes this return that is estimated via the optimal action-value function,

$$\mathcal{Q}^{\star}(s,a) = max_{\pi}\mathbb{E}[R_t|s_t = s, a_t = a, \pi], \tag{18}$$

$$\mathcal{Q}^{\star}_{i+1}(s,a) = \mathbb{E}_{s\prime\sim\mathcal{E}}[r + \gamma max_{a\prime}\mathcal{Q}^{\star}_{i}(s\prime,a\prime)|s,a], \tag{19}$$

updated iteratively using the **Bellman equation** (19). In practice, a function approximator is used to estimate the action-value function with θ as parameters: $\mathcal{Q}(s,a;\theta) \approx \mathcal{Q}^{\star}(s,a)$.

In deep reinforcement learning, that function approximator is a deep (convolutional) neural network called **Q-network**, following [14]. In this RL scheme, such a network is trained in a supervised way by minimizing the following loss function at each iteration i :

$$L_i(\theta_i) = \mathbb{E}_{s,a}\Big[(y_i - \mathcal{Q}(s,a;\theta_i))^2\Big], \tag{20}$$

$$y_i = \mathbb{E}_{s\prime\sim\mathcal{E}}\Big[r + \gamma max_{a\prime}\mathcal{Q}(s\prime,a\prime;\theta_{i-1})\Big] \approx \mathcal{Q}^{\star}(s,a), \tag{21}$$

where an estimation of the optimal action-value function y_i (21) is used. As stated earlier, many issues come about when introducing neural networks as function approximator in a RL scheme. In order to alleviate on those issues, some peculiarity within the algorithm can be stressed out:

- The first things that stands out is the use of an **experience replay** where experiences are stored and then pooled over into a replay memory that is used at each time step. The

main goals of that experience replay are, firstly, to allow a greater data efficiency and, secondly, to stabilize the learning process so that the updates of the Q-network weights θ can be done safely.

- The second interesting point regards the way that experience is learned from: in this algorithm, learning is not done directly from the consecutive samples in which case a strong correlations between samples would prevent the Q-network to converge on global minima. Instead, learning is done in an *off-policy* manner in order to decorrelate the experiences from which learning is done. That is to say that the replay memory is sampled randomly to create batches from which the Q-network learns in a supervised way.

2.2 RELATED WORKS

Given the context in which we wish to deploy that deep reinforcement learning framework, to wit the context of a swarm of robots that operate each in parallel of the others, great concern has been attached to that concept of asynchronism/parallelism. In deep reinforcement learning, luckily, it has been proven to be very beneficent to both reduce the computational time required to train a neural network-based controller and address the dynamic stability issue of such a training by enabling a decorrelation of the data from which the learning agent learns, following [24] and [15]. In the followings, we mainly take inspiration from the state-of-the-art that consists of the *Advantage Asynchronous Actor Critic (A3C)* algorithm from [15] and the *Deep Deterministic Policy Gradient (DDPG)* algorithm from [16].

There is one last think that we have not addressed yet, to wit the shape of the action space $\mathcal{A}$ and the way the policy π is dealt with. Firstly, the action space can be discrete or continuous and, depending on that charasteric, the architecture of our neural network will change and our working hypothesis will also vary. In our robotic framework, we will need to tackle continuous action spaces, so in the remaining of this report $\mathcal{A}$ will always denote a continuous action space. Secondly, the policy π can be modeled as a stochastic policy or a deterministic policy: the effectiveness of deterministic policies has been proven in [16] when the action space is continuous.

It is important to notice that, while the *A3C* algorithm is an stochastic policy-based *"on-policy"* algorithm (the attribution of the rewards is greatly dependent to the applied policy during the phase where the agent experiences and explores the environment), the DDPG algorithm is a deterministic policy-based *"off-policy"* algorithm (based on an actor-critic scheme, the attribution of the rewards is decorrelated from the actual way through which the agent experiences the environment, thanks to the use of an experience replay buffer).

2.3 CURRENT APPROACH

We have chosen to tackle a mix of those two approaches explained in [15] and [16] in the form of the algorithm explained as follows, entitled *Deep Deterministic Policy Gradient-Based Asynchronous Actor-Critic (DDPG-BA2C)*.

It is a multi-threaded algorithm where the main thread exploits the current policy π parameterized by the weights θ_{actor} and populate an experience replay buffer R while the other

// Randomly initialize the actor's and critic's weights θ_{actor} and θ_{critic};
// Initialize the target actor's and target critic's weights $\theta^{target}_{actor} = \theta_{actor}$ and $\theta^{target}_{critic} = \theta_{critic}$;
// Initialize replay buffer R;
for *episode* = 1,M **do**
    Receive initial observation state s_1;
    for t = 1,$T_{episode}$ **do**
        Select action $a_t = \pi(s_t; \theta_{actor}) + \epsilon$ according to the current policy and some exploration noise ϵ;
        Execute action a_t and observe reward r_t and new state s_{t+1};
        Compute action $a_{t+1} = \pi(s_{t+1}; \theta_{actor})$ Store transition $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ in R;
    end
    Perform a given number of epochs of the actor-learner function;
end

Figure 22.: Algorithm DDPG-BA2C's pseudocode for the main actor loop.

threads update those weights in an asynchronous/parallel and decorrelated fashion by sampling random mini-batches of experiences from the experience replay buffer. The same as in *DDPG*, two target networks, parameterized by the weights θ^{target}_{actor} and θ^{target}_{critic}, are used to stabilize the learning process. The target networks are updated towards the learning networks at the frequency with a momentum τ in a soft-update fashion. Figure 22 details the algorithm followed by the main loop, while Fig. 23 details the threads' one.

Validation experiments of the *DDPG-BA2C* algorithm have been undertaken and the results are displayed in *Appendix B*.

// Assume globally shared parameter for the actor's and critic's weights θ_{actor} and θ_{critic};
// Assume globally shared parameters for the target actor's and target critic's weights θ^{target}_{actor} and θ^{target}_{critic};
// Wait for the replay buffer R to be populated;
for *epoch* = 1,N_{epoch} **do**
  for *replay* = 1,N_{replay} **do**
    Sample randomly $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ from R;
    Compute $\mathcal{Q}(s_t, a_t; \theta_{critic})$;
    Compute $a'_{t+1} = \pi(s_{t+1}; \theta^{target}_{actor})$;
    Compute $\mathcal{Q}(s_{t+1}, a'_{t+1}; \theta^{target}_{critic})$;
    Compute the error $\delta^{target} = r_t + \gamma \mathcal{Q}(s_{t+1}, a'_{t+1}; \theta^{target}_{critic}) - \mathcal{Q}(s_t, a_t; \theta_{critic})$;
    Accumulate the gradient that minimizes the error δ^{target} with respect to the critic weigths, with (s_t, a_t) as input :

$$d\theta_{critic} = d\theta_{critic} + \nabla_{\theta_{critic}} \delta^{target}|_{s=s_t, a=a_t}$$

    Accumulate the actor weights' gradient in the direction of the sampled policy gradient (that maximizes the state-action value) :

$$d\theta_{actor} = d\theta_{actor} + \nabla_a \mathcal{Q}(s, a; \theta_{critic})|_{s=s_t, a=a_t} \nabla_{\theta_{actor}} \pi(s; \theta_{actor})|_{s=s_t}$$

  end
  Perform asynchronous update of θ_{critic} and θ_{actor} by using, respectively, $d\theta_{critic}$ and $d\theta_{actor}$;
  Update the target networks :

$$\theta^{target}_{critic} = \tau \theta^{target}_{critic} + (1 - \tau)\theta_{critic}$$
$$\theta^{target}_{actor} = \tau \theta^{target}_{actor} + (1 - \tau)\theta_{actor}$$

end

Figure 23.: Algorithm DDPG-BA2C's pseudocode for each actor-learner thread.

3

DEEP REINFORCEMENT LEARNING AND NONLINEAR SYSTEM-BASED COUPLED CONTROLLER

3.1 RELATED WORKS

To our knowledge, there were only a few approaches which tried to combine biologically-plausible feedback controller design with conventional feedback controller design. The main works that we want to emphasize are the work of Kawato et al. [25] and Gomi et al. [26]. Following the identification of neural pathways in the human central nervous systems that are relevant to motor control tasks , Kawato et al. devised an architecture that tries to replicate those pathways. Eventually they refined it into the *Feedback-Error Learning* scheme that consist of three components, the main component being a neural network that tries to learn the inverse model of a controlled object thanks to an error signal that comes from a conventional feedback controller whose goal is to control the controlled object with regards to a desired trajectory specified as input to the whole system. The main purpose of this architecture is to solve the problem of translating the error signal from the task space (or observation space, to wit the space in which the sensors operate) to the motor command space (the space in which the actuators are controlled). Gomi et al. [26] carried on that path by expanding the Feedback-Error Learning architecture into to two different approaches, to wit the Inverse Dynamics Model Learning and the Nonlinear Regulator Learning schemes.

3.2 ARCHITECTURE

Although the previous related works are not based on the reinforcement learning paradigm, the described architectures have a lot in common with our current reinforcement learning based approach as it will be emphasized in the followings, mainly with regards to the feedback error signal.

We will denote as *Reinforced NonLinear-Based Neural Controller (RNLBNC)* our devised architecture that consists of a nonlinear system-based controller (NLC) coupled in parallel to a neural network controller (NC), as depicted in Fig. 24. Thus, our work has more in common with the Inverse Dynamics Model Learning scheme in the sense that no reference trajectory is being given to the reinforcement learning agent, but only a feedback-error in place of the reward (or penalty) provided by the environment. In each scheme, there are parallel information flows that are summed up (actions a_t^{NLC}, from the NLC, and a_t^{NC} from the NC) in the motor command space to create the motor control inputs, or action a_t.

It is important to emphasize that, in both our architecture and Gomi's Inverse Dynamics Model Learning architecture, the neural network controller learns from a motor command

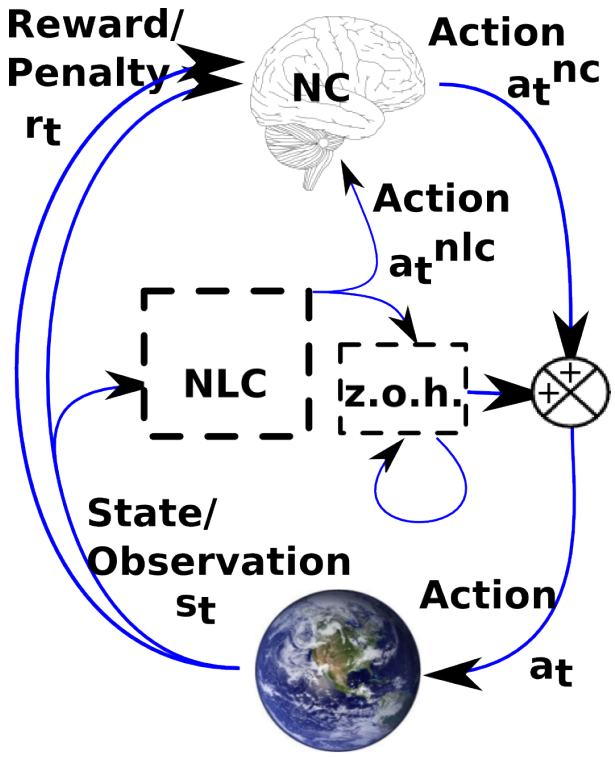

Figure 24.: Reinforced NonLinear-Based Neural Controller's architecture.

space-grounded error signal. The discrepancy, though, stands in the fact that we rely on a reinforcement learning paradigm that provides a reward/penalty signal to the NC, here the RL agent, in order to evaluate its efficiency with regards to our system's goal.

3.3 REWARD SHAPING

In reinforcement learning, given a goal that we would like the RL agent to achieve, it is can be difficult to devise an efficient reward/penalty function that will guide the agent in its learning and accomplishment. Thus, reward function shaping is a critical aspect of reinforcement learning. Here, thanks to our reliance on a nonlinear system-based controller that encodes our goal in a behavior, as explained previously in this work, we can devise an automatic reward shaping process.

Considering a nonlinear/complex system in its autonomous form, with t as time,

$$\dot{x}(t) = f(x(t)),$$

where $x \in \mathbb{R}^n$ with $n \in \mathbb{N}$ and f a nonlinear function. We assume that our goal is achieved for some equilibrium point x_0 such that $f(x_0) = 0$. Thus, we propose to define the reward function r_t as the inverse of the sum of this system's energy E_t and a given distance measure of the current state $x(t)$ from the desired equilibrium point x_0,

$$E_t = \frac{1}{2}\dot{x}(t)^T\dot{x}(t), \tag{22}$$

$$r_t = \frac{1}{E_t + \alpha\|x(t) - x_0\|}, \tag{23}$$

where $\|\|$ is a given norm, for instance the euclidean norm, and α is a regularization factor. The importance of the energy-based term relies in the fact that we want the agent to learn a

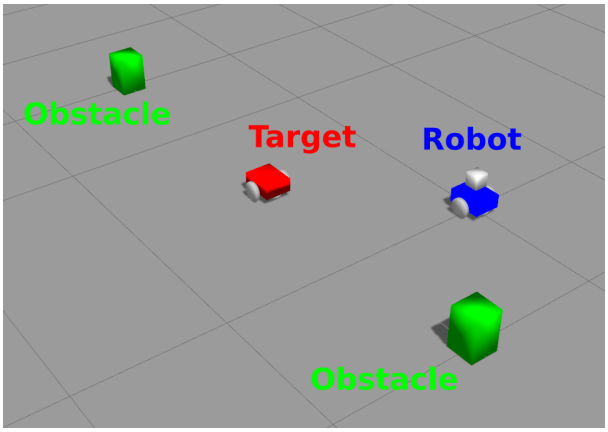

Figure 25.: Experimental setup of the proof of concept experiment.

behavior that is achieved by the efficient minimization of the nonlinear system's energy. In case of a nonlinear system with multiple equilibrium points, it is of the utmost importance to emphasize in the reward function the desired equilibrium point.

3.4 SIMULATIONS

3.4.1 *Experimental setup*

As a proof of concept of the *RNLBNC*, we apply the *DDPG-BA2C* algorithm to a neural network controller, coupled with the obstacle avoidance-augmented target-encircling control law presented in Sec. 4.2. As first experiment, we restrain ourselves to the problem of one robot trying to encircle a fixed target while avoiding fixed obstacles, as presented in Fig. 25.

Moreover, learning directly from pixels an optimal policy requires too much time, so we decided to give to the NC as observation the real state of the system instead. This state is described as

$$s_t = \begin{bmatrix} r(t) \\ \theta(t) \\ \phi(t) \\ r_{obs}(t) \\ \theta_{obs}(t) \end{bmatrix},$$

where, at each time step t, $r(t)$ is the distance from the robot to the target, $r_{obs}(t)$ is the distance from the robot to the closest obstacle, and the angles $\theta(t)$ and $\theta_{obs}(t)$ are defined as presented in Fig. 1 and 14, respectively. $\phi(t)$ is the angular position of the robot in the global reference frame centered on the target. As specified in [14], feeding the neural network with concatenations of consecutive states, instead of each state at a time, improves the results of the learned policy by enabling the neural network to actually develop velocity-based features. So, in the following experiments, we feed stacks of the 4 last states at each time step to the neural network.

We evaluate the *RNLBNC* against the nonlinear system-based controller alone (NLC), and the reinforcement learning agent that consist of the same neural network controller (NC) architecture as the one used in the *RNLBNC*. It is important to notice that the position of the robot

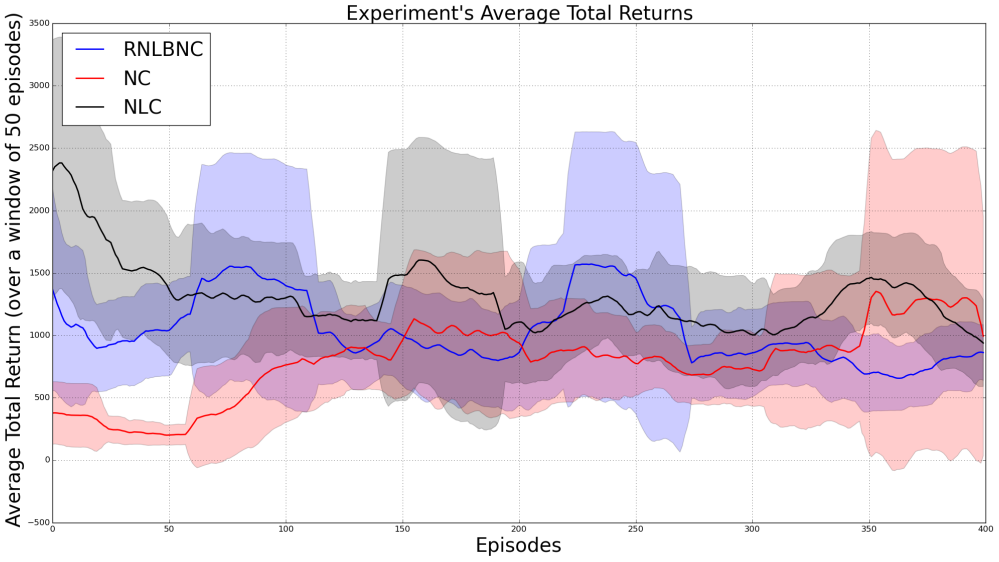

Figure 26.: Comparison of *RNLBNC*, *NLC*, and *NC* over 400 episodes of 2000 steps each on the target-encircling task with one robot and multiple obstacles. The y-axis shows the total return $R = \sum_t r_t$ for each episode. The higher the total return, the better is the performance.

at the beginning of each episode is randomly chosen. It makes the task way much more difficult than if the robot was launched on a desirable position, for instance on its desired target-encircling trajectory.

3.4.2 *Results*

In this experiments, the instances of the *DDPG-BA2C* algorithm of the *RNLBNC* and the *NC* were using, respectively, 8 and 4 threads with mini-batches of size 64 and 32. They both used the same learning rate $\lambda = 1e-4$ to update the weights, and the same soft-update factor $\tau = 1e-3$ to update the target networks.

Figure 26 shows the average of the total return against the episodes while Fig. 27 shows it in decibel in order to emphasize more clearly the differences. At each episode, *NC* improves towards the baseline *NLC*. As expected, *RNLBNC* performs fairly the same as the baseline *NLC*, right from the beginning. Thus, we show that *RNLBNC* as great potential for applications in critical situations where we cannot afford to start with low-performances.

The source code for this experiment is open-source and available here : `https://github.com/Near32/GazeboRL/tree/energy-based-reward`.

3.5 DISCUSSION

We have shown a proof of concept for the *Reinforced NonLinear-Based Neural Controller* scheme that is still to be improved. It still needs to be validated in a less controlled situation where the neural networks are not fed the state of the system but rather the actual sensor signal.

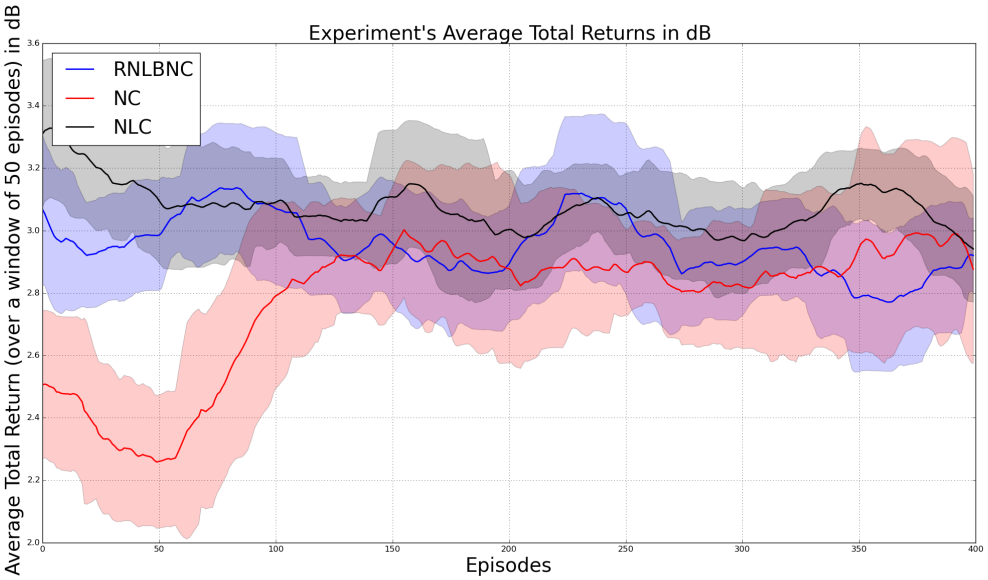

Figure 27.: Comparison of *RNLBNC*, *NLC*, and *NC* over 400 episodes of 2000 steps each on the target-encircling task with one robot and multiple obstacles. The y-axis shows the total return $R = \sum_t r_t$ for each episode, scaled in decibel. The higher the total return, the better is the performance.

This way, in our current swarm robotic framework, the neural network would need to be equipped with a convolutionnal network since the actual sensor signal is a video stream. We leave that validation and the deployment of the RNLBNC algorithm for the swarm of robots to subsequent works.

Since the RNLBNC scheme enables us to have a near-optimal policy in the place of the NLC, right from the start, we could devise an unsupervised auxiliary task in which the NC would try to learn to imitate the NLC first, as proposed in [27]. This task could enable the current architecture to achieve better data efficiency.

4

CONCLUSION

We have described the *Reinforced NonLinear-Based Neural Controller* scheme and a proof of concept experiment in which a near-optimal behavior was acknowledge right from the start of the learning process, thus enabling such architecture to be deployed in critical situations with limited risks. This architecture combines adaptability features, thanks to its deep reinforcement learning-based parts, and a robust nonlinear system-based controller that is already showing robust performances on the given task. The learning phase of this system is no longer impairing the performances and thanks to its adaptability features it will be able to improve over time.

Appendices

A

DISTRIBUTED MODEL

As explained previously, each robot i needs to estimate three main quantities, to wit r_i, θ_i and ψ_i. In order to do so, in a decentralized way, each robot needs to be equipped with a camera-typed sensor that enables it to observe its surroundings and thus track the target, that needs to be encircled, and its fellow neighbour robots to decide with which one to couples, at the oscillator level. For each robot of the swarm to be able to compute its control law, it is required to solve the well-studied problem of *Detection And Tracking of Moving Objects (DATMO)* as each robot needs to know where the target is and where its fellow neighbours are. Once the surroundings has been analyzed, it remains to reframe everything into a reference frame that suits the concerned robot as explained previously. Let us detail how to solve the DATMO problem in the followings.

A.1 HARDWARE TO SOLVE THE DATMO PROBLEM :

We have chosen to equipped every robot with a *Raspberry Pi* 3 as onboard computer and its associated camera mounted on a rotating mount with a fisheye lens in order to enable the robot to quickly have a sense of its surroundings. This choice of hardware was done by default as the usual omnidirectional lenses were not affordable.

The current state of the final robot is shown in Fig. 28 with its two-wheeled robot double-deckered base.

A.2 SOFTWARE TO SOLVE THE DATMO PROBLEM :

In the following simulations, a simple colorimetric filter is implemented in order to detect and track the neighbour robots and the target. But, in reality, a colorimetric filter has little chances to succeed. So, multiple solutions have been investigated and we will try to explains those in the following.

A.2.1 *Classical Computer Vision-based Solution : Planar-based Solution*

In In real environments, opposed to simulations, we would have to deal with multiple issues such as:

- lightning-based changes of the features,

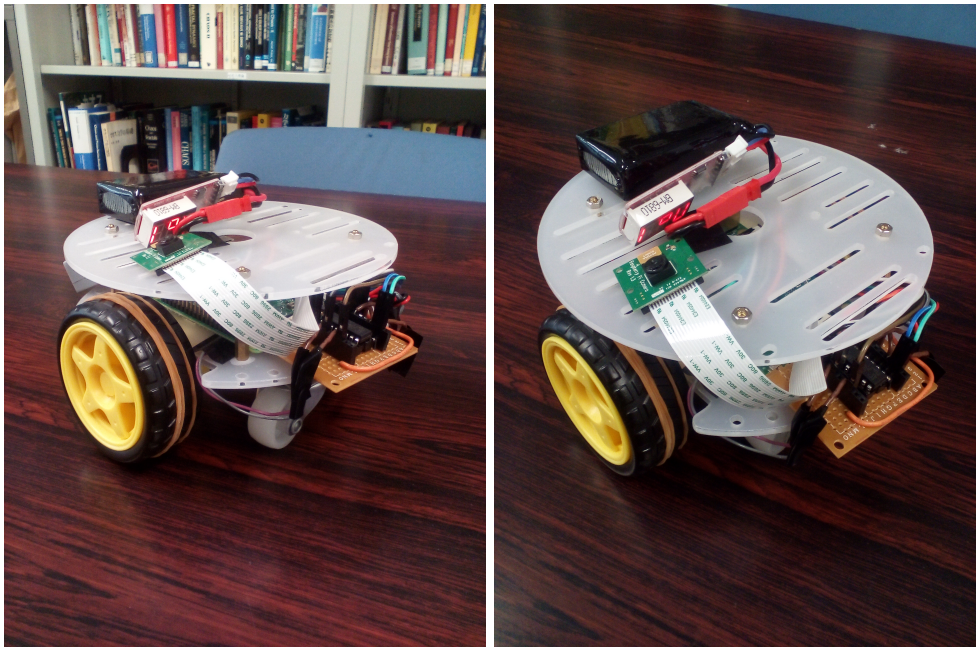

Figure 28.: Camera mount (left), fisheye lens (middle) and current state of the robot (right).

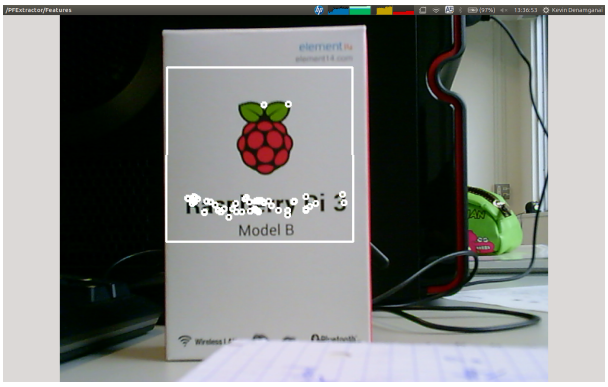

Figure 29.: Example of a recognition of the Raspberry Pi logo relying on the planar-based solution implemented.

- recognition rate too low due to the poor quality of the picture when used with the omnidirectional lens.
- execution time too long when used with more powerful feature point descriptors...

This first method aim at recognizing feature points of given objects and also the planar shape that those feature points describe. We plan to apply that method to the recognition of neighbour robots in a real environments. The geometrical constraints added into the recognition process are thought to yields better accuracy without adding on the time complexity cost. Figure 29 shows the algorithm at work with intent to recognize the Raspberry Pi logo.

A.2.2 *Qualitative Evaluation of the Planar-based Solution*

We evaluate the planar-based solution on a video streaming where the object we seek to recognized is moved from right to left and so forth. The object distance from the camera is

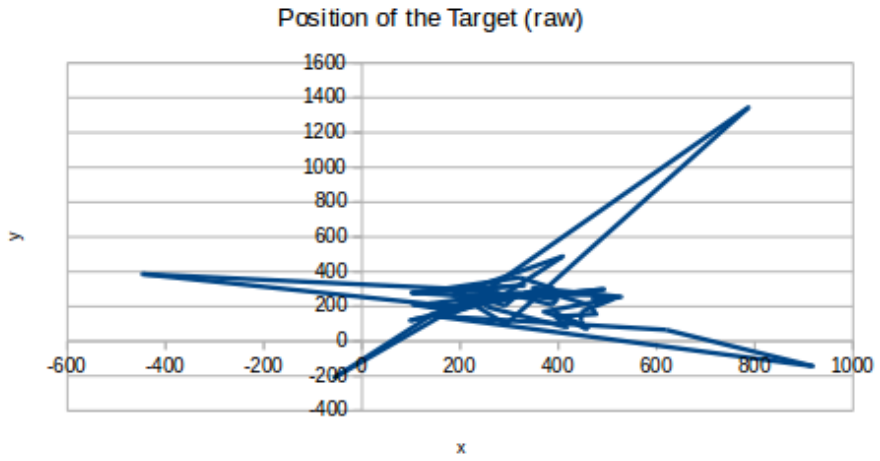

Figure 30.: Raw output of the planar-based recognition algorithm.

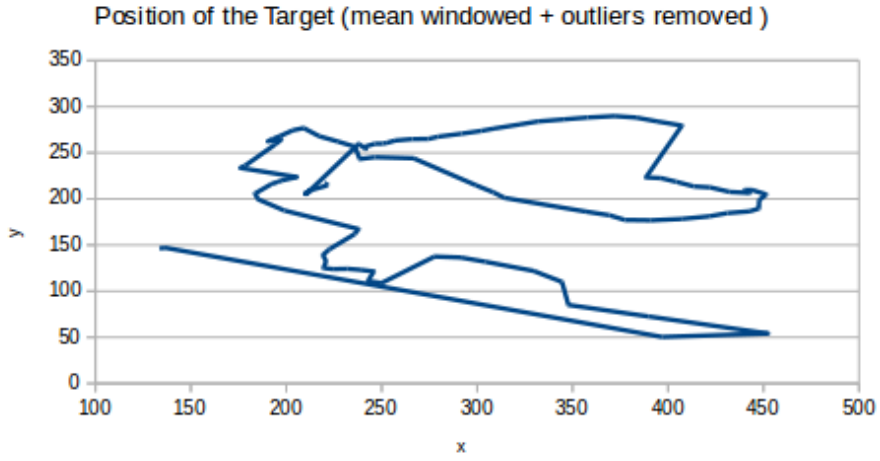

Figure 31.: Filtered output of the planar-based recognition algorithm.

approximately 30 centimeters. Figure 30 shows the output of the algorithm without any filtering applied, whereas Fig. 31 shows results after mean filtering over a time-window of 10 estimations and outliers rejection filtering.

A.2.3 *Quantitative Static Evaluation of the Planar-based Solution*

Provided a video stream that contains the object we seek to recognize without any movement, we evaluate the planar-based solution algorithm quantitatively with regards to the maximal standard deviation of the noise induced by the algorithm on its output. Figure 32 shows the output of the algorithm, after the appropriate filtering, using a standard pinhole camera, while Fig. 33 shows the results with the omnidirectional camera we set out to use on the robots. Given the very poor results we have for the omnidirectional camera, we set out to use a super-resolution algorithm in order to enhance the quality of the image retrieved from the omnidirectional camera. Using this pipeline, Fig. 34 shows the output of the algorithm. *Table 1* compares the aforementioned algorithm with regards to the maximal standard deviation

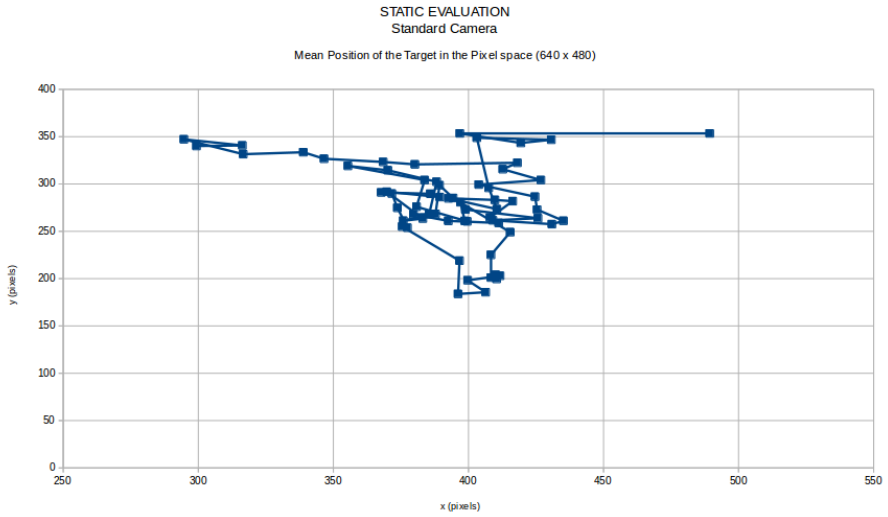

Figure 32.: Output of the planar-based solution algorithm with a standard pinhole camera on the static evaluation.

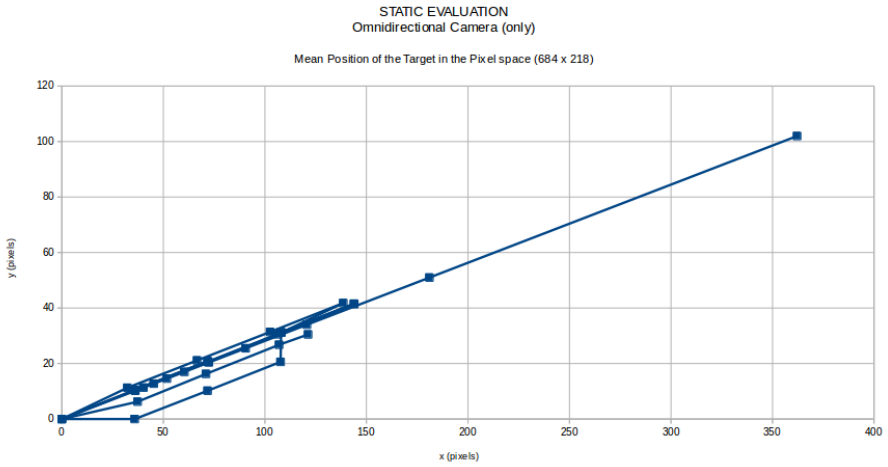

Figure 33.: Output of the planar-based solution algorithm with a omnidirectional camera on the static evaluation.

on the X and Y axises estimated from the experiments.

	Standard Camera	Omnidirectional Camera	Omnidirectional Camera + Super-resolution
X	20.3%	44.1%	21.0%
Y	15.2%	39.4%	16.5%

Table 1.: Maximal Standard Deviation of the output noise in each settings with respect to each axis.

In the end, averaged over the two axises, we have roughly 18% of maximal error on the recognition of an object at a distance of 30 centimeters of the cameras. Those results are not trustworthy enough to build a whole system over it.

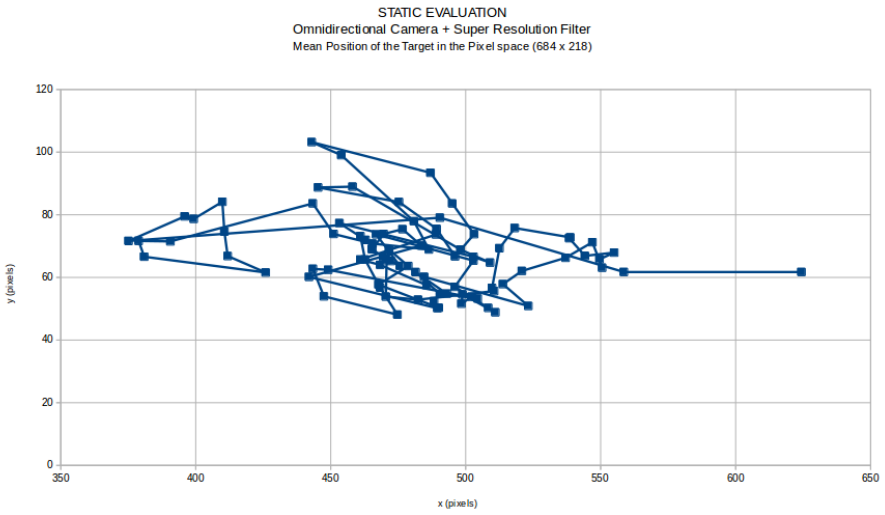

Figure 34.: Output of the planar-based solution algorithm with a omnidirectional camera coupled with a super-resolution filter on the static evaluation.

A.2.4 *Deep Learning-based Solution :*

Given the poor results of the previous approach, we set out to investigate some deep learning-based solutions. In this peculiar field, two tasks are being tackled in order to solve our neighbour robots recognition problem :

- Recognition/Classification Task
  (**"Is it a robot or a chair ?"**)
- Bounding Box Regression Task
  (**"Where is it located in the picture ?"**)

To that end, two state-of-the-art architectures were hoped to be investigated:

- **Faster R-CNN** [28],
- **Spatial Transformer Network** [29].

Faster R-CNN is the result of multiple incremental innovations in order to make the solution run in real-time under a given hardware set. Basically, this algorithm encompasses a convolutionnal network whose output feature maps are used for both region of interest proposal estimation and classification of those regions of interest. The whole pipeline is an end-to-end solution for solving the aforementioned tasks.

While this previous architecture clearly separate the flow of information into two paths, one for each task, as seen in Fig. 35, the **Spatial Transformer Network** embeds the two tasks in one and only path. The feature maps extracted are restrained to the main region of interest where the object we seek to identify is residing with the highest probability. That is to say that this architecture can only recognize and identify one object at a time. In order for it recognize and identify multiple objects, we need to make multiple pass on the same image while making sure that we blurred out (or completely removed) the previous region of interest that it has outputted on the previous pass. Unfortunately, because of time constraints, we ended up testing only the **Spatial Transformer Network** architecture.

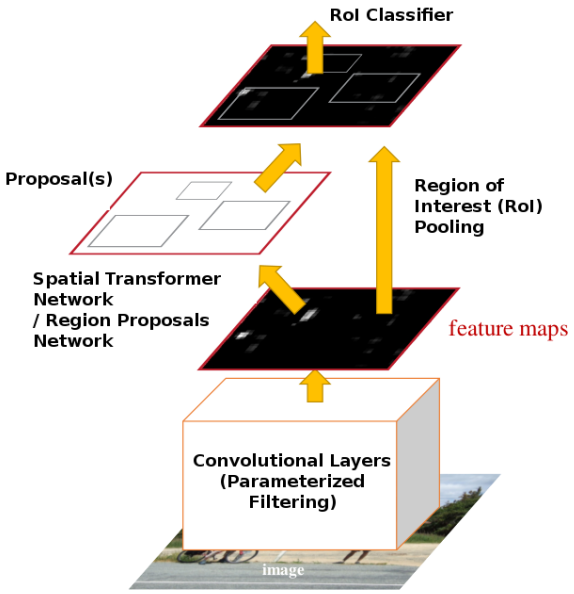

Figure 35.: Workflow of the **Faster R-CNN** and **Spatial Transformer Network** algorithms.

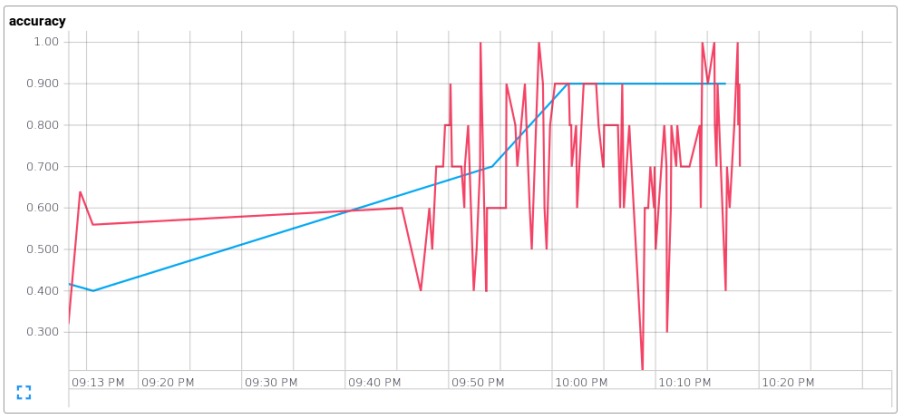

Figure 36.: Test (blue) & train (red) accuracies of a {reluconvx2-16-32+STN+relumaxpoolconvx1-64+128xFC} neural network.

As always in deep learning, specifically with supervised learning, we need to rely on a dataset. We have created and artificial dataset that comprised the following information :

- real indoor scene images taken from the **IMAGENET** database [30],
- labels/bounding box of the object we seek to identify.

We denote it as "artificial" because of the fact that the object we seek to identify was not originally part of the scene but artificially pasted on the scene after having undergone some random projection transformations, following the method developed in [31]. The object that was pasted on the images was the Hyundai and Subaru logos.

Figure 36 shows the accuracy of the **Spatial Transformer Network** during training on the recognition task. $90 \pm 10\%$ of accuracy on test dataset is achieved. More extensive test might be necessary though, but the lack of time made us stop here.

In the end, this approach yield a rather comfortable accuracy of $90 \pm 10\%$ for the recognition task and its result on the bounding box regression task seem to show good recall while the precision is not good nor bad, qualitatively speaking.

B

DDPG-BA2C ALGORITHM: VALIDATION EXAMPLE

B.1 PROBLEM FORMULATION

In order to validate the efficiency of the algorithm *DDPG-BA2C*, let us test it on a well-known problem, the **Cart Pole Problem**. The goal is to stabilize the center of mass upward ($\theta = \pi$) by pulling or pushing the cart, as shown in Fig. 37. The agent thus controls the force $F \in [-100; 100]$. Fig. 38, 39 and 40 shows the evolution of the policy.

B.2 EFFECT OF REWARD SHAPING

Along with the efficiency of the algorithm, another point can be highlighted. It regards the chosen reward functions. Previously we had:

$$R_t = -(\theta(t) - \pi)^2 \tag{24}$$

This reward function only cares about the position of the center of mass that should be upward. Instead, if we consider to penalize the effort applied by the agent and also try to constraint the cart at the origin $x = 0$, we end up with the following new reward function :

$$\begin{aligned} R_t &= -(\frac{\theta(t) - \pi}{\pi})^2 - \beta x^2(t) - \gamma |F| \\ \beta &= 5\,,\, \gamma = 5 \end{aligned} \tag{25}$$

where the action at time step t is $a_t = F$.

Under that new reward function, learning has been accelerated. As we can see, by episode 40, already a near-optimal policy is exhibited, as displayed in Fig. 41.

The whole source code is open source and available here : `https://github.com/Near32/NeuralNetworks/tree/RL`.

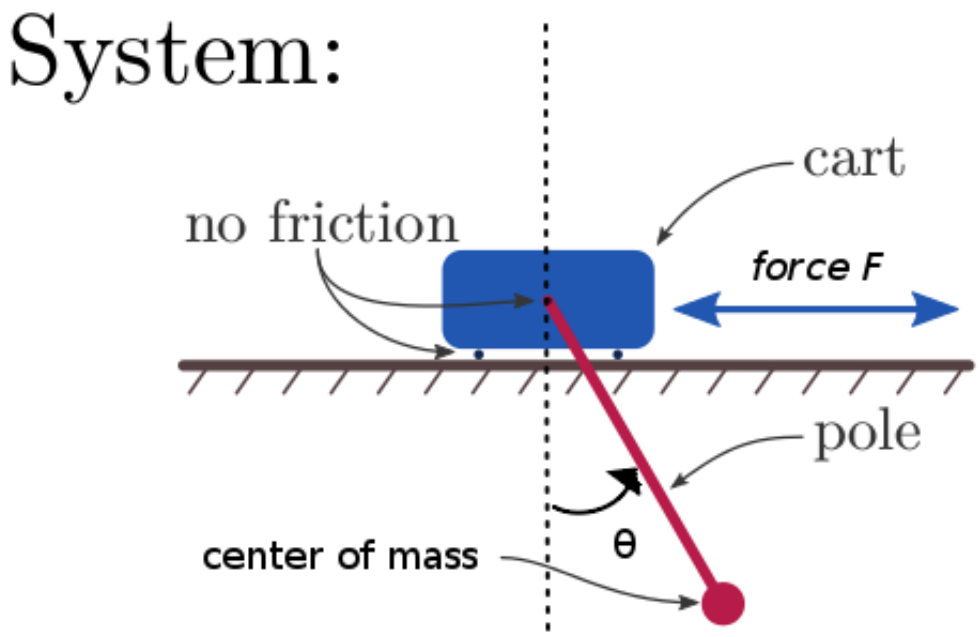

Figure 37.: Framework of the Cart-pole Problem.

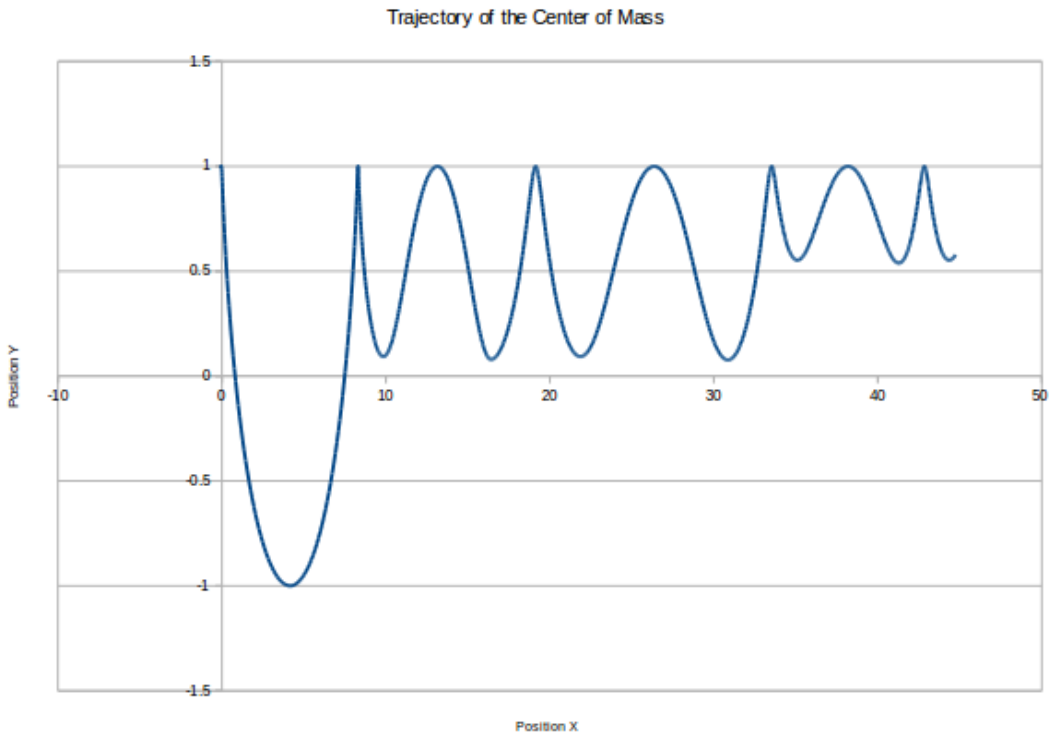

Figure 38.: Trajectory of the center of mass in the X-Y plane, by episode 600.

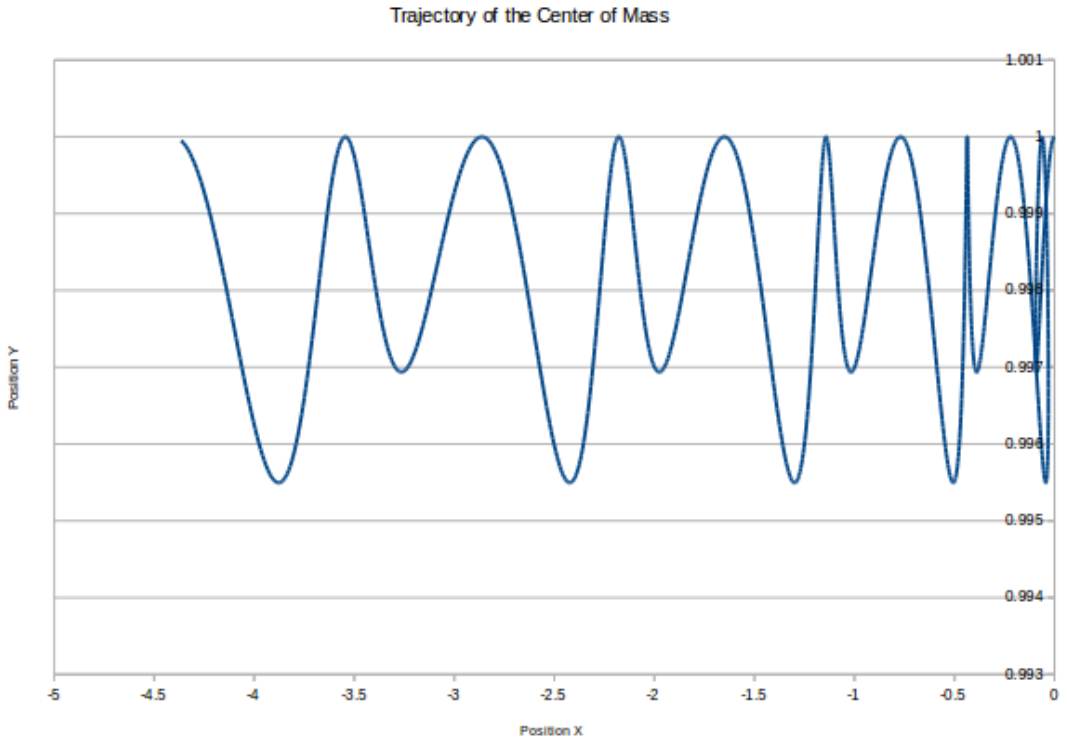

Figure 39.: Trajectory of the center of mass in the X-Y plane, by episode 1000.

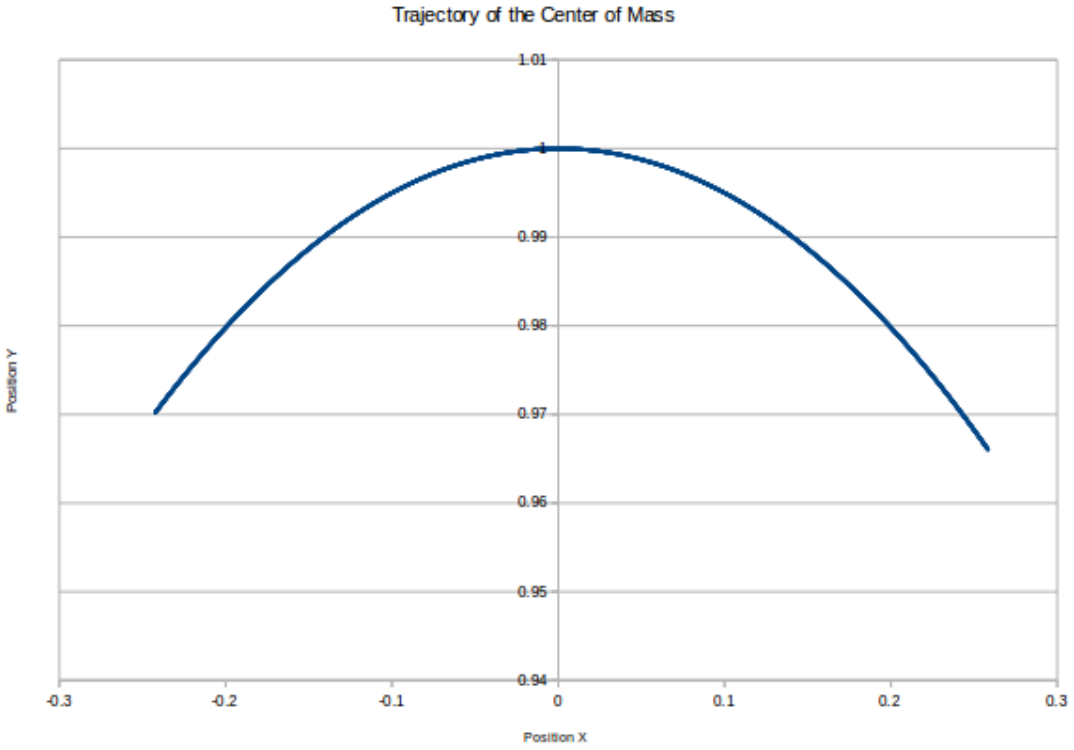

Figure 40.: Trajectory of the center of mass in the X-Y plane, by episode 2000.

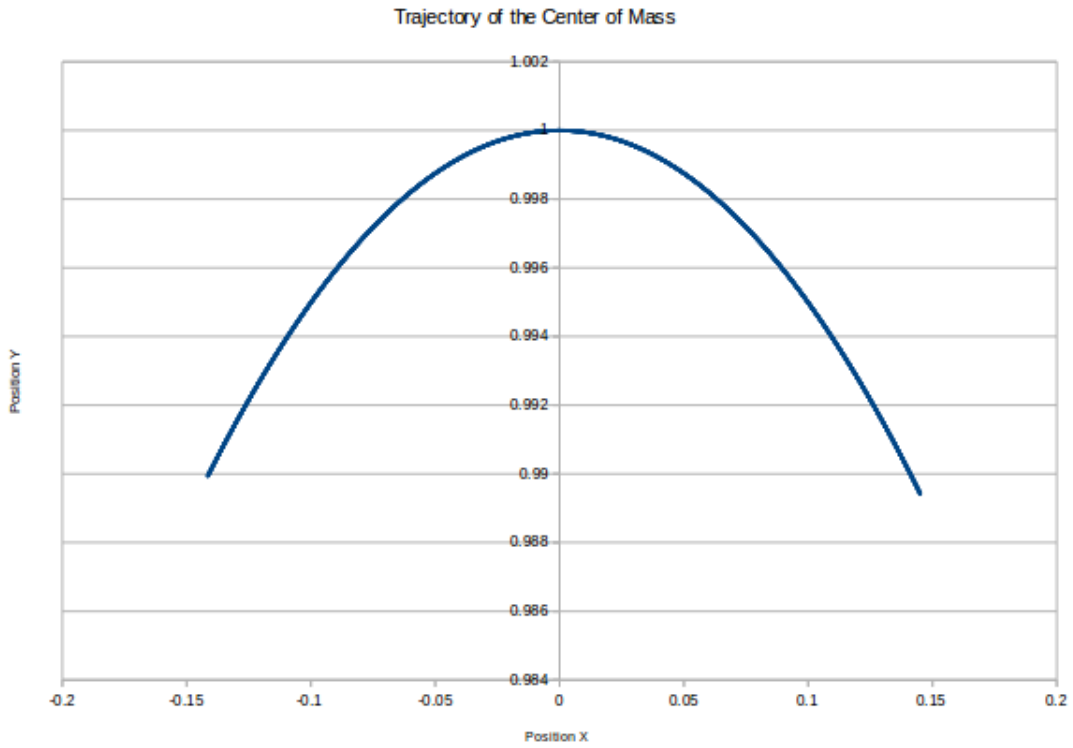

Figure 41.: Trajectory of the center of mass in the X-Y plane, by episode 40, using efficient reward-shaping.

ACKNOWLEDGEMENTS

Without the assistance and support from friends, colleagues and professors at Osaka Prefecture University and elsewhere, this thesis would not have been possible. There are a lot of people I want to thank for their help in attaining my Master course.

Firstly, I would like to express my sincere gratitude to my supervisor Professor Keiji Konishi for giving me the opportunity to pursue my Master degree at Osaka Prefecture University, and for his availability, guidance and support during my year and half as a Master student in Osaka, Japan. Furthermore, I would like to thank specially Professor Naoyuki Hara as well as every member of my laboratory group for their support and care.
I also want to thank people of my former graduate school, the ENSEA, without whom I would not have had the opportunity to pursue this Master degree.

Finally, I would like to extend my thanks to each person who has had positive influences on my life during this past year and a half, making it a very instructive and interesting experience.

BIBLIOGRAPHY

[1] Rafael Fierro, Aveek Das, John Spletzer, Joel Esposito, Vijay Kumar, James P. Ostrowski, George Pappas, Camillo J. Taylor, Yerang Hur, Rajeev Alur, Insup Lee, Greg Grudic, and Ben Southall. A framework and architecture for multi-robot coordination. *The International Journal of Robotics Research*, 21(10-11):977–995, 2002.

[2] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Auton. Robots*, 8(3):345–383, June 2000.

[3] Erkin Bahceci, Onur Soysal, and Erol Sahin. A review: Pattern formation and adaptation in multi-robot systems. Technical Report CMU-RI-TR-03-43, Robotics Institute, Pittsburgh, PA, October 2003.

[4] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. In *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, volume 3, pages 2953–2958 vol.3, Dec 2002.

[5] Nima Moshtagh, Ali Jadbabaie, and Kostas Daniilidis. Vision-based control laws for distributed flocking of nonholonomic agents.

[6] R. Olfati-Saber. Flocking for multi-agent dynamic systems: algorithms and theory. *IEEE Transactions on Automatic Control*, 51(3):401–420, March 2006.

[7] William M. Spears. Distributed adaptive swarm for obstacle avoidance. *International Journal of Intelligent Computing and Cybernetics*, 2(4):644–671, 11 2009.

[8] Jamie Snape, Stephen J. Guy, Jur Van Den Berg, and Dinesh Manocha. *Smooth coordination and navigation for multiple differential-drive robots*, volume 79 of *Springer Tracts in Advanced Robotics*, pages 601–613. Springer Verlag, Germany, 2014.

[9] Dimitra Panagou, Dusan M. Stipanovic, and Petros G. Voulgaris. Multi-objective control for multi-agent systems using lyapunov-like barrier functions. In *CDC*, 2013.

[10] Urs Borrmann, Li Wang, Aaron D Ames, and Magnus Egerstedt. Control barrier certificates for safe swarm behavior. 2015.

[11] C. Gros. *Complex and Adaptative Dynamical Systems*. Springer-Verlag, (2008).

[12] F.A. Rodrigues, T.K.DM. Peron, P. Ji, and J. Kurths. The Kuramoto model in complex networks. *Physics Reports, vol.610, pp.1-98*, (2015).

[13] R.S. Sutton and A.G. Barto. Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[15] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[16] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[17] Kevin Denamganai, Tadashi Nakamura, Naoyuki Hara, and Keiji Konishi. Obstacle avoidance control law for two-wheeled mobile robots controlled by oscillators. *Proceedings of the 61st Annual Conference of the Institute of Systems, Control and Information Engineers (ISCIE)*, pages 221–4, 2017.

[18] Kevin Denamganai, Tadashi Nakamura, Naoyuki Hara, and Keiji Konishi. Coupled Kuramoto oscillator-based control laws for both formation and obstacle avoidance control of two-wheeled mobile robots. *IEICE Technical Report*, 117(121):87–91, 2017.

[19] Naoyuki Hara, Hideki Kokame, and Keiji Konishi. Circular periodic motion generation for mobile robots using limit cycle systems. In *American Control Conference (ACC), 2010*, pages 4271–4276. IEEE, 2010.

[20] H. Kokame K. Yamada, N. Hara and K. Konishi. Implementation and experimental validation of circular periodic motion generation for mobile robots using limit cycle systems. In *SICE Annual Conference*, page 3391–3394, 2010.

[21] Y. Takarada N. Hara and K. Konishi. Coupled nonlinear oscillators with application to mobile robot formation control. In *SICE Annual Conference*, pages 2174–2177, 2013.

[22] N. Hara M. Tsukiji and K. Konishi. Experimental evaluation of formation control of mobile robot group by coupled oscillators. In *SICE Annual Conference*, pages 301–306, 2014.

[23] Tadashi Nakamura, Miyuki Tsukiji, Naoyuki Hara, and Keiji Konishi. Stability analysis of mobile robot formations based on synchronization of coupled oscillators. *IFAC-PapersOnLine*, 49(22):187–191, 2016.

[24] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296, 2015.

[25] Mitsuo Kawato. Feedback-error-learning neural network for supervised motor learning. In *Advanced Neural Computers*, pages 365 – 372. 1990.

[26] Hiroaki Gomi and Mitsuo Kawato. Neural network control for a closed-loop system using feedback-error-learning. *Neural Networks*, (6):933–946, 1993.

[27] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[28] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[29] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial Transformer Network. *Nips*, pages 1–14, 2015.

[30] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[31] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. {FlowNet}: {L}earning Optical Flow with Convolutional Networks. *{IEEE} International Conference on Computer Vision*, pages 2758–2766, 2015.

PUBLICATIONS

The contents of this thesis have been submitted in conference proceedings. The links between these publications and each chapter of the thesis are listed below.

- Chapter II.3 : (1), (2)
- Chapter II.4 : (1), (2)

Presentations in domestic conferences:

1. Kevin Denamganai, Tadashi Nakamura, Naoyuki Hara and Keiji Konishi, "Obstacle avoidance control law for two-wheeled mobile robots controlled by oscillators", Proceedings of the 61st Annual Conference of the Institute of Systems, Control and Information Engineers (ISCIE), 221-4, 2017.

2. Kevin Denamganai, Tadashi Nakamura, Naoyuki Hara and Keiji Konishi, "Coupled Kuramoto oscillator-based control laws for both formation and obstacle avoidance control of two-wheeled mobile robots", IEICE Technical Report, vol. 117, no. 121, NLP2017-44, pp. 87-91, 2017.