ACTION GENERALIZATION IN HUMANOID ROBOTS THROUGH ARTIFICIAL INTELLIGENCE WITH LEARNING FROM DEMONSTRATION

By
Raúl Fernández Fernández

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in

Electrical Engineering, Electronics and Automation

At

Universidad Carlos III de Madrid

Advisors
Carlos Balaguer Bernaldo de Quirós
Juan Carlos González Víctores

Tutor
Juan Carlos González Víctores

September, 2021

Some rights reserved. This work is licensed under the CreativeCommons
Attribution-Non Commercial-Non Derivatives 3.0 (CC BY-NC-ND 3.0).

“I saved a life. My own. Am I a hero? I really can’t say.
But, yes.”

Michael Scott, The Office.

AKNOWLEDGEMENTS

Although this thesis is written in English allow me to use Spanish for this. This is the mother tongue of most of the people I want to thank.

Primero de todo quiero darle las gracias a Juan y Carlos, mis tutores. Sin ellos esta tesis no sería posible, es tan suya como mía. Gracias a Carlos por confiar en mi desde el primer momento y ser uno de los mejores "jefes" que he tenido. Gracias a Juan por todo su apoyo y esfuerzo a lo largo de la tesis, gracias por dejarme escaparme de vez en cuando de viaje, y gracias por ser mi amigo. El resto de agradecimientos están ordenados por orden cronológico. Tranquilos, os quiero a todos igual, solo que a algunos os quiero antes que a otros.

Gracias a mis padres, Mama y Papa, por estar siempre ahí, quererme tanto, y enseñarme tantas cosas buenas. Gracias a mi hermano, Alex, por lo bien que lo hemos pasado y por lo mucho que nos hemos divertido juntos. Gracias a mis abuelos, Jovino, Laudelina, Erundina y Cueto, por todos los buenos momentos. Gracias al resto de mi familia, Hector, Paula, Aitana, Adam, Bea, Henar, Esther, Jose Antonio... por haberme querido tanto. Gracias a Javi, Isa y Celia, por haber sido como mi segunda familia. Gracias a Nuria, Armando y Mario, por todas las risas juntos. Gracias a la gente de Peñerudes, Borja, Martin, Pablo, Jose, Adrián, Verónica, Izán, Sergio, Eva, Dorita, Vicente, Nosa... por ser tan buena gente y por todas las veces que me fuisteis a buscar a casa. Gracias a todos mis profesores, JJ, Candido, Corsino, Adela, Susana, Cristina... por haber sido tan buenos ejemplos a seguir. Gracias a mis compañeros de clase, Yahre, Lara, Sergio,

Carlos, el otro Sergio, Abraham, Noelia, Lucía, Alejandro, Luis, David, la otra Lara, Juanpa, Roberto, Jinsop, Andreu, Aitor, Bruno, Ana, Fabián, el otro Fabián, Cristina, el otro Alejandro, Rafa, Diego, Pelayo... por hacer el día a día más llevadero. Gracias a Pepe y a Javi por haber sido siempre tan buenos amigos. Gracias a Guille por descubrirme tantos hobbies nuevos para los que no tengo ni tiempo ni dinero. Gracias al resto del Concilio, Dani, Domi, Pif, Alex y Jairo, por todas las risas que nos echamos juntos. Gracias al resto de mis amigos, Tania, Cris, Darío, Varito, Rafa, Raquel, Mar, Abadin, Adriana, Karla, Inna, Alicia, Adriana, Lucas, Sandra, Jaime, Cris, Hector, Inés, Chechu, Adriana, Noe, Noli, Indra, Juanki, Miriam, Henar, Lara, Mata, Adrian, Meg, Rato, Marta, Rocío, Mendefu, Olga, Oscar, Pam, Cesar, Rebe, Guille, Mario... por tantas cosas buenas.

Por último quería hacer un agradecimiento especial a toda la gente del Roboticslab y la UC3M, Clara, Alejandra, Kike, Sergio, Javier, Miguel, David, Javi, Santiago, Edu, Sonia, Santi, Ramón, Concha, Alvaro, Marcos, Bartek, Jorge, Dorin, Carba, Pavel, Lisbeth, Luis, Pedro, Edwin, Fernando, el otro Luis, Dolores, Mohamed, Alvaro, Hanno, Esther, el otro Jorge, Juan, Marina, Raúl, Juanjo, la otra Marina, Sara, Angela, Fernando, Silvia, Jennifer, Ana, Andrea... y todos los que se me olvidan pero estoy agradecido igual. Gracias a todos por haber hecho de esta experiencia algo tan positivo. Quería también agradecer a mis compañeros del C13, a Olaya por todos los momentos haciendo el idiota y lo bien que nos lo hemos pasado juntos, a Elisabeth por ser la mejor compañera de piso del mundo, a David por robarme los cojines, a Raúl por compartir tantas cosas conmigo, a Alice por ser una tía tan guay, a Noé por no enfadarte cuando nos metiamos con Andalucía, a Alvaro por ser tan buena persona y a Marcos por ser tan buen rival al Mario Party. Thanks to the people at INRIA, Claudio, Marco, Alberto, Adam, Abdul, Alexander, Amaia, Josu... you made me feel very welcome. Por último, gracias a toda la gente que se me olvida por todo lo demás.

PUBLISHED AND SUBMITTED CONTENT

The most relevant published/submitted international journals that are included in this thesis are presented in the following list.

1. **Raul Fernandez-Fernandez**, Juan G. Victores, Jennifer J. Gago, David Estevez, and Carlos Balaguer. Neural Policy Style Transfer. *Cognitive Systems Research*, 2021. **JCR (2020): Q3**. (Under revision: original submitted Jun-2020, revision submitted April-2021)

   * This source is wholly included in Chapters 5 and 6 of this thesis.

2. **Raul Fernandez-Fernandez**, Juan G. Victores, David Estevez, and Carlos Balaguer. Real Evaluations Tractability using Continuous Goal-Directed Actions in Smart City Applications. *Sensors*, 18(11):3818, 2018. ISSN 1424-8220. doi: 10.3390/s18113818. **JCR: Q1**. (Published)

   * This source is wholly included in Chapters 2 and 3 of this thesis.

The most relevant published peer-reviewed conference papers that are included in this thesis are presented in the following list.

1. **Raul Fernandez-Fernandez**, Juan G. Victores, David Estevez, and Carlos Balaguer. Robot Imitation through Vision, Kinesthetic and Force Features with Online Adaptation to Changing Environments. *In IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1-9. IEEE, 2018. doi: 10.1109/IROS.2018.8593724.

   * This source is wholly included in Chapters 2 and 4 of this thesis.

2. **Raul Fernandez-Fernandez**, David Estevez, Juan G. Victores, and Carlos Balaguer. Improving CGDA execution through Genetic Algorithms incorporating Spatial and Velocity constraints. *In IEEE Int. Conf. on Autonomous Robot Systems and Competitions (ICARSC)*, pages 290-295. IEEE, 2017. doi: 10.1109/icarsc.2017.7964090.

   * This source is wholly included in Chapters 2 and 3 of this thesis.

3. **Raul Fernandez-Fernandez**, David Estevez, Juan G. Victores, and Carlos Balaguer. Reducing the number of evaluations required for CGDA execution through Particle Swarm Optimization methods. In *IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 284-289. IEEE, 2017. doi: 10.1109/ICARSC.2017.7964089.

   * This source is wholly included in Chapters 2 and 3 of this thesis.

4. **Raul Fernandez-Fernandez**, Juan G. Victores, and Carlos Balaguer. New Trends and Challenges in the Automatic Generation of New Tasks for Humanoid Robots. *In RoboCity16 Open Conference on Future Trends in Robotics*, pages 169-176, 2016. ISBN 978-84-608-8452-1.

   * This source is partially included in the Chapter 1 of this thesis.

The material from these sources included in this thesis is not singled out with typographic means and references.

OTHER RESEARCH MERITS

This list includes other scientific contributions that have not been included in this thesis but have been developed and published in parallel.

1. David Estevez, Juan G. Victores, **Raul Fernandez-Fernandez**, and Carlos Balaguer. Enabling garment-agnostic laundry tasks for a Robot

Household Companion. *Robotics and Autonomous Systems*, 2020. ISSN 0921-8890. doi: 10.1016/j.robot.2019.103330. **JCR: Q2**.

2. **Raul Fernandez-Fernandez**, Juan G. Victores, David Estevez, and Carlos Balaguer. Quick, Stat!: A Statistical Analysis of the Quick, Draw! Dataset. *In 10th EUROSIM Congress on Modelling and Simulation*. ARGESIM, 2019. doi: 10.11128/arep.58.

3. David Estevez, Juan G. Victores, **Raul Fernandez-Fernandez**, and Carlos Balaguer. Towards Clothes Hanging via Cloth Simulation and Deep Convolutional Networks. *In 10th EUROSIM Congress on Modelling and Simulation*. ARGESIM, 2019. doi: 10.11128/arep.58.

4. David Estevez, Juan G. Victores, **Raul Fernandez-Fernandez**, and Carlos Balaguer. Robotic Ironing with 3D Perception and Force/ Torque Feedback in Household Environments. *In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017. doi: 10.1109/IROS.2017.8206556.

5. David Estevez, **Raul Fernandez-Fernandez**, Juan G. Victores, and Carlos Balaguer. Improving and evaluating robotic garment unfolding: A garment-agnostic approach. *In IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 284-289. IEEE, 2017. doi: 10.1109/ICARSC.2017.7964077.

   * Awarded Best Student Paper of the conference.

6. David Estevez, **Raul Fernandez-Fernandez**, Juan G. Victores, and Carlos Balaguer. Robotic ironing with a humanoid robot using human tools. *In IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 134-139. IEEE, 2017. doi: 10.1109/ICARSC.2017.7964065.

ABSTRACT

Action Generalization is the ability to adapt an action to different contexts and environments. In humans, this ability is taken for granted. Robots are yet far from achieving the human level of Action Generalization. Current robotic frameworks are limited frameworks that are only able to work in the small range of contexts and environments for which they were programmed. One of the reasons why we do not have a robot in our house yet is because every house is different.

In this thesis, two different approaches to improve the Action Generalization capabilities of robots are proposed. First, a study of different methods to improve the performance of the Continuous Goal-Directed Actions framework within highly dynamic real world environments is presented. Continuous Goal-Directed Actions is a Learning from Demonstration framework based on the idea of encoding actions as the effects these actions produce on the environment. No robot kinematic information is required for the encoding of actions. This improves the generalization capabilities of robots by solving the correspondence problem. This problem is related to the execution of the same action with different kinematics.

The second approach is the proposition of the Neural Policy Style Transfer framework. The goal of this framework is to achieve Action Generalization by providing the robot the ability to introduce Styles within robotic actions. This allows the robot to adapt one action to different contexts with the introduction of different Styles. Neural Style Transfer was originally

proposed as a way to perform Style Transfer between images. Neural Policy Style Transfer proposes the introduction of Neural Style Transfer within robotic actions.

The structure of this document was designed with the goal of depicting the continuous research work that this thesis has been. Every time a new approach is proposed, the reasons why this was considered the best new step based on the experimental results obtained are provided. Each approach can be studied separately and, at the same time, they are presented as part of the larger research project from which they are part. Solving the problem of Action Generalization is currently a too ambitious goal for any single research project. The goal of this thesis is to make finding this solution one step closer.

Raúl Fernández Fernández

ACTION GENERALIZATION IN HUMANOID ROBOTS THROUGH ARTIFICIAL INTELLIGENCE WITH LEARNING FROM DEMONSTRATION

LIST OF ABBREVIATIONS

AFFG	**A**daptive **F**uzzy **F**itness **G**ranulation
CBR	**C**ase **B**ased **R**easoning
CGDA	**C**ontinuous **G**oal-**D**irected **A**ctions
DDPG	**D**eep **D**eterministic **P**olicy **G**radient
DoF	**D**egrees **o**f **F**reedom
DQN	**D**eep **Q**-**N**etwork
DRL	**D**eep **R**einforcement **L**earning
DTW	**D**ynamic **T**ime **W**arping
FA	**F**itness **A**pproximation
FI	**F**itness **I**nheritance
FTE	**F**ull **T**rajectory **E**volution
GAN	**G**enerative **A**dversarial **N**etwork
IE	**I**ndividual **E**volution
IET	**I**ncrementally **E**volved **T**rajectories
IRL	**I**nverse **R**einforcement **L**earning
NPST	**N**eural **P**olicy **S**tyle **T**ransfer
NPST3	**N**eural **P**olicy **S**tyle **T**ransfer TD**3**
PSO	**P**article **S**warm **O**ptimization
RIT	**R**eal **I**teration **T**ime
RT	**R**obot **T**hreshold
SST	**S**teady **S**tate **T**ournament
TD3	**T**win **D**elayed **D**eep **D**eterministic Policy Gradient
OET	**O**nline **E**volved **T**rajectories

TABLE OF CONTENTS

AKNOWLEDGEMENTS V
PUBLISHED AND SUBMITTED CONTENT VII
ABSTRACT XI
LIST OF ABBREVIATIONS XIII
LIST OF FIGURES XIX
LIST OF TABLES XXV

I Introduction **1**
Chapter One: Introduction 3
- 1.1 An Introduction to Action Generalization 4
- 1.2 The goal of this thesis 7
  - 1.2.1 Improving Continuous Goal-Directed Actions (CGDA) 8
  - 1.2.2 NPST: Introducing Style Transfer in robot actions 10
- 1.3 Relevant concepts and elements 11
  - 1.3.1 Elements of Learning from Demonstration 12
  - 1.3.2 Elements of Evolutionary Algorithms 12
  - 1.3.3 Elements of Reinforcement Learning 13

Conclusions of Part I: Introduction 16

II Continuous Goal-Directed Actions **17**
Chapter Two: Continuous Goal-Directed Actions 19
- 2.1 The CGDA framework 20
- 2.2 Generalization 22
- 2.3 Recognition 24
- 2.4 Execution 26
  - 2.4.1 FTE: Full Trajectory Evolution 26

2.4.2 IE: Individual Evolution 27
2.4.3 IET: Incrementally Evolved Trajectories 28
2.5 Conclusions 29

Chapter Three: Reducing evaluations in CGDA 31
3.1 Introducing Approximations 32
3.2 Particle Swarm Optimization in CGDA 33
3.2.1 Particle Swarm Optimization 34
3.2.2 Adaptive Fuzzy Fitness Granulation PSO 34
3.2.3 Fitness Inheritance PSO 35
3.3 Introducing Constraints 36
3.4 Approximations and Constraints in CGDA 39
3.5 Experiments 40
3.6 Evolutionary Approximation Results 43
3.6.1 Wax Results 43
3.6.2 Paint Results 45
3.7 Constrained Genetic Algorithms Results 47
3.7.1 Wax Results 47
3.7.2 Paint Results 50
3.8 Conclusions 52

Chapter Four: Online Evolved Trajectories 55
4.1 Online Evolutionary Strategies 56
4.2 The Online Evolved Trajectories algorithm 58
4.2.1 Perception Step 58
4.2.2 Localization Step 59
4.3 Experiments 59
4.4 Results 61
4.4.1 Paint Results 62
4.4.2 Iron Results 64
4.5 Conclusions 64

Conclusions of Part II: Continuous Goal-Directed Actions 69

III Neural Policy Style Transfer (NPST) 69

Chapter Five: Reinforcement Learning and Style Transfer 71
5.1 Reinforcement Learning 71
5.1.1 Q-Learning 72
5.1.2 Deep Reinforcement Learning 73
5.1.3 Continuous action spaces 74

5.2 Style Transfer: Content and Style 75
    5.2.1 Neural Style Transfer 76
    5.2.2 Conclusions 77

Chapter Six: NPST in discrete action spaces 79
6.1 Inverse Reinforcement Learning 81
6.2 Neural Policy Style Transfer 83
6.3 Experiments 86
    6.3.1 Catch-ball Game Experiment 86
    6.3.2 Grid-world Paint Experiment 89
6.4 Results 91
    6.4.1 Catch-ball Game Results 91
    6.4.2 Grid-world Paint Results 93
6.5 Conclusions 94

Chapter Seven: NPST3 in continuous action spaces 97
7.1 Framework 98
    7.1.1 Inputs 99
    7.1.2 Autoencoder Network: loss network 100
    7.1.3 Constraints 100
    7.1.4 TD3 Policy Network: execution network 101
    7.1.5 Outputs 103
7.2 Training 103
7.3 Experiments 104
    7.3.1 Setup 105
    7.3.2 Questionnaire and Subjects 106
    7.3.3 Results 106
7.4 Conclusions 108

Conclusions of Part III: Neural Policy Style Transfer 111

IV Conclusions **111**

Chapter Eight: Conclusions 113
8.1 Main Conclusions and Results 113
8.2 Innovations 115
8.3 Future Work 117
    8.3.1 Proposed Enhancements 117
    8.3.2 Proposed Applications 118

References 123

LIST OF FIGURES

1 A CBR workflow example. The process is not sequential and all stages may occur at any time. 5

2 User demonstration of the painting action using the humanoid robot TEO within the CGDA framework. 8

3 Style Transfer example using the algorithm proposed by Gatys et al. (1) with an image of TEO and the Starry Night of Vincent Van Gogh. 10

4 Continuous Goal-Directed Actions (CGDA) framework diagram. 21

5 Plot of a 1-dimensional feature action generalization. Colorized lines depict different demonstrations. The black line is the generalized feature trajectory obtained from all the demonstrations. 23

6 Case example of applying DTW between two trajectories. The two trajectories used in this example are the same, but with different time executions. 25

7 Simulated execution of the “paint” action using the humanoid robot TEO and Openrave. 41

8 “Wax” Evolutionary Approximation results: total number of required evaluations for the “wax” action execution as a function of the intermediate goals defined in the feature trajectory. 44

9 “Wax” Evolutionary Approximation results: feature trajectories obtained with the following algorithms: A) SST, B) PSO, C) AFFG-PSO and D) FI-PSO. The green trajectory depicts the generalized feature trajectory encoding the action. The blue trajectory is the feature trajectory obtained with the execution stage of CGDA. 44

10 “Paint” Evolutionary Approximation results: results as a function of each intermediate goal defined by the generalized trajectory. At the top, the total number of required evaluations. At the bottom, total percentage of painted wall. 46

11 “Wax” spatial constraint experiment results: total number of required evaluations for the “wax” action execution as a function of the intermediate goals defined in the feature trajectory. 48

12 “Wax” velocity constraint experiment results: total number of required evaluations for the “wax” action execution as a function of the intermediate goals defined in the feature trajectory. 49

13 “Paint” spatial constraint experiment results: total number of required evaluations for the “paint” action execution as a function of the intermediate goals defined in the feature trajectory. 50

14 “Paint” velocity constraint experiment results: total number of required evaluations for the “paint” action execution as a function of the intermediate goals defined in the feature trajectory. 51

15 The OET algorithm allows introducing changes in the environment during execution. In the image, a collaborative execution of the“paint” action is depicted. 56

16 Resulting trajectories for the demonstrations of the “paint” action. The orange thick line depicts the result of executing the Gaussian Mixture Regression method ($K = 7, T = 600$) using these demonstrations. 62
17 “Paint” OET experiments results: generalized feature trajectory for the “paint” action compared to the features trajectories obtained for each of the algorithms. 63
18 Resulting trajectories for the demonstrations of the “iron” action. The orange thick line depicts the result of executing the Gaussian Mixture Regression method ($K = 5, T = 150$) using these demonstrations. 65

19 Deep neural network architecture similar to the one proposed by Mnih et al. (2) for the introduction of DRL in Atari games. 73

20 Grid-world paint scenario demonstration setup. 80
21 Neural Policy Style Transfer (NPST) framework. Three DQN are proposed with the same architecture. The content DQN ($\mathbb{C}$) is trained using the reward function extracted from the content action demonstrations. The Style DQN ($\mathbb{S}$) is trained using the reward function extracted from the Style action demonstrations. The Generated DQN ($\mathbb{G}$) is obtained using the output layer Q-values of $\mathbb{C}$ and the full weights of $\mathbb{S}$. 84
22 Screenshots obtained during the NPST execution with the “nervous” Style (top) and “fall” Style (bottom). Red bars are the output of the Content network $\mathbb{C}$. Blue bars are the output of the Generated policy $\mathbb{G}$. 92

23 Agent heat-map. From left to right: Content, Style and Generated policies. “Nervous” Style (top) and “fall” Style (bottom). Pixels with warms colors have been recurrently visited by the agent. Cold colors represent less visited pixels. The depicted scale is the cumulative result of the 10 different executions of the NPST algorithm. 93

24 NPST3 base idea: human demonstrations define the Style while the Content can be defined online via teleoperation. 98

25 NPST3 framework: The input of the Autoencoder network are the Style, Content and Generated actions. The loss obtained with the Autoencoder (L_{st}) is added to the loss defined by the Constraints (L_p, L_{ep}, L_v) to obtain the total loss L. The inverse of this loss is used as the reward for training the TD3 Policy Network. In addition to this reward, the input of the TD3 network are the Content and Generated actions. This network defines a control policy that outputs a 3D Cartesian velocity to the robot end-effector. The velocity command is executed and the new position of the robot is obtained. The Generated trajectory is updated using this new position. The Content trajectory is updated using the information provided by the user. This Content trajectory can be defined offline or online via teleoperation. 99

26 Cartesian trajectories obtained in the experiments. The top trajectory is the selected Content action. For each pair of trajectories, the left trajectory corresponds to one of the four selected Styles: A) Angry, B) Happy, C) Calm and D) Sad. The trajectory in the right corresponds to the generated action obtained with the NPST3 framework. Style trajectories are defined via demonstrations with the right hand of a volunteer using a Vicon mo-cap system. Warm colors depict a higher velocity value, while cold colors depict lower velocities. 105

27 NPST3 questionnaire first part results. Each chart depicts the answers given for one of the videos. The title on the top right of the chart depicts the original emotion transferred to the shown video. The labels on the X axis are the emotions written by the volunteers. The Y axis is the number of times that emotion was used to describe the video by a volunteer. 107

28 NPST3 questionnaire second part results. Each row corresponds to a different video. Each column corresponds to the Style selected by the volunteers. Each cell contains the number of times a given Style was selected by a volunteer. All the videos were presented to all the volunteers. 108

LIST OF TABLES

3 “Wax” Evolutionary Approximation results: average results after 50 repetitions of the “wax” action for each of the proposed algorithms. 43

4 “Paint” Evolutionary Approximation results: average results after 100 repetitions of the “paint” action for each of the proposed algorithms. 45

5 “Wax” spatial constraint experiment results: average results after 50 repetitions of the “wax” action for each of the dilatation values. 47

6 “Wax” velocity constraint experiment results: average results after 50 repetitions of the “wax” action for each of the velocity constraint values. 49

7 “Paint” spatial constraint experiment results: average results after 100 repetitions of the “paint” action for each of the spatial constraint values. 50

8 “Paint” velocity constraint experiment results: average results after 100 repetitions of the “paint” action for each of the velocity constraint values. 51

9 “Paint” OET experiment results: average and standard deviation results after 3 repetitions of the “paint” action for each of the proposed strategies. 63

10 “Iron” OET experiments results: average and standard deviation results after 3 repetitions of the “iron” action for each of the proposed strategies. 64

11 Hyper-parameters defined for the Catch-ball game. 88
12 Hyper-parameters defined for the Grid-world paint scenario. 90
13 Catch-ball game experimental results. 91
14 Grid-world paint experimental results. 93

15 Training hyper-parameters for the NPST3 algorithm. 104

Part I

Introduction

1. INTRODUCTION

The idea of learning is conceived as the acquisition of a set of rules or steps that allow performing a certain action. In the case of human learning, the assumption is that once you have learned how to perform an action, as long as some requisites are present, you can repeat that action in many different contexts. For example, when you were a child and your parents taught you how to make the bed, from that day, you were supposed to make the bed every day. Telling your parents excuses in the fashion of “These blankets are bigger than the others, so I do not know how to do this anymore”, or “You taught me how to do the bed in summer; I do not know what a quilt is or how to use it” or “The bed is slightly moved to the left with respect to where it was when you taught me” were normally not accepted by your parents. You were trapped in the fact of being able to make the bed and therefore had to do it. You were even supposed to be able to make beds that were different from the one you had in your house. In humans, the ability to adapt a learned action is taken for granted.

In robotics, this ability is called Action Generalization, and is one of the hardest problems existing in the robotic community. Robots are very limited by differences between the context where they learned the action and where they have to execute it. In this thesis, a study of how to deal with this problem is presented. First, Continuous Goal-Directed Actions (CGDA) developed by Morante et al. (3) is presented, studied and improved as an implementation of a learning and Action Generalization framework. In this thesis, CGDA is improved to work with real robots in real environments.

In addition, following the breakthrough of Deep Learning methods, Neural Policy Style Transfer (NPST) –an application of state of the art deep machine learning techniques– is proposed. This framework introduces Deep Reinforcement Learning (DRL) for the robust execution of trajectories generated using Style Transfer techniques. Finally, Neural Policy Style Transfer TD3 (NPST3), a more advanced version of NPST working with continuous action spaces and introducing Autoencoders to extract the Content and Style, is proposed.

The rest of this chapter provides a background to the reader to understand the following relevant topics presented in this thesis: what is Action Generalization; what is the scope of this thesis; what is CGDA and what are the improvements proposed; what is the idea behind NPST; an introduction to the related relevant concepts; and finally some conclusions which sum up and review all this information.

1.1 An Introduction to Action Generalization

The idea of Action Generalization was preceded and inspired by the idea of Inductive Learning. The idea of Inductive learning was initially introduced in 1739 by Hume (4). Inductive learning is based on a method of reasoning that consists on extracting, from concrete pieces of experience, general rules than can be applied to a wider range of contexts (5). An example of an inductive argument can therefore be something like the following one extracted from Herms (6):

`All swans I have seen in the past have been white.`
`It follows that the next swan I see will be white.`

Following this, a more pragmatic example closer to the goals of this thesis can be derived from this idea:

`All painters I have seen have used a brush.`

`It follows that if I have to paint I will use a brush.`

Following this idea, the computer science community came up with a method to use past experiences to solve new problems called Case Based Reasoning (CBR). The idea of CBR was to use the knowledge of past experiences to partially or fully solve new problems similar to ones already faced (7).

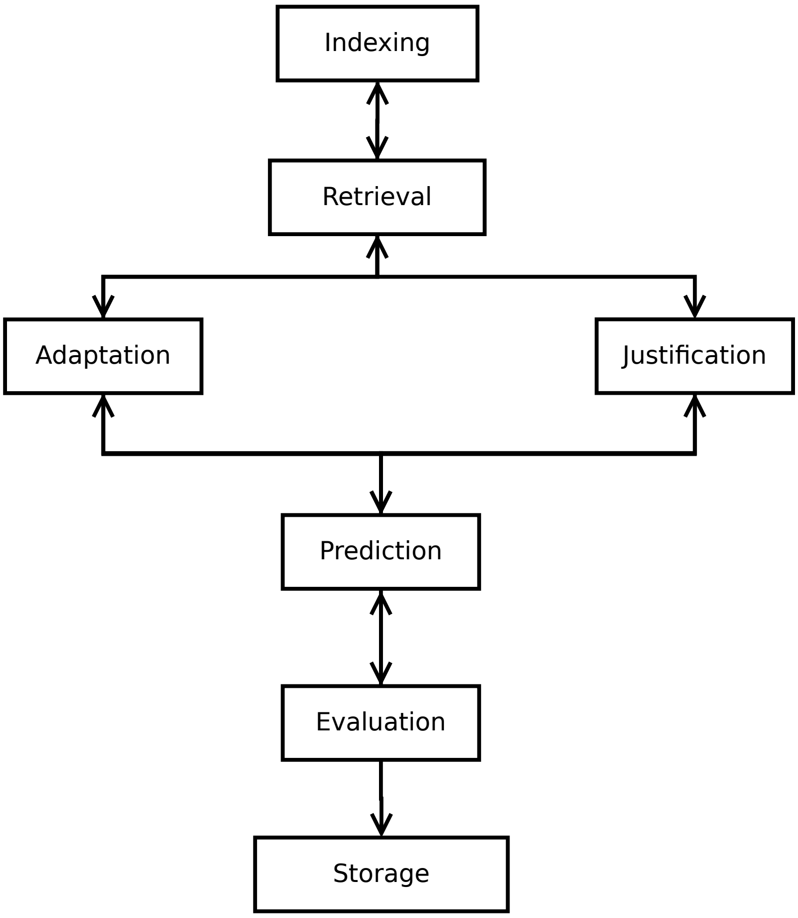
Indexing
Retrieval
Adaptation
Justification
Prediction
Evaluation
Storage

Figure 1: A CBR workflow example. The process is not sequential and all stages may occur at any time.

Fig.1 depicts the stages typically presented in CBR. These stages are:

- Indexing stage: stored experiences are labeled to be stored and later retrieved.

- Retrieval stage: experiences with labels related to the new problem are selected from the database.
- Adaptation stage: old experiences are modified to adapt to the new problem.
- Prediction and Justification stage: a simulation of what is going to happen is performed as an initial evaluation of the reliability of the generated solution.
- Evaluation stage: the action is executed in the real world and an evaluation is performed about the success of the generated solution.
- Storage stage: the new learned solution is stored as a new experience in the database.

CBR, however, has not had a large impact in the robotics community. One of the reasons may be that although conceptually CBR works, its implementation in a real robotic system presents too many problems that have yet to be solved.

More recent approaches using ideas similar to the ones presented in CBR have been more successful at dealing with Action Generalization problems in robotics. In Learning from Demonstration, Action Generalization is presented in the correspondence problem. The correspondence problem appears in the context of how to teach a robot to perform certain actions while the demonstrator and robot kinematics may be completely different. As defined by Argall et al. (8), the correspondence problem is present in the mapping between the demonstrator and the learner, and how to allow the transfer of information between them. In Calinon et al. (9), the authors used Gaussian distributions to encode simple manipulation actions from demonstrations, that can be generalized to be executed in robots with different kinematics.

Semantic learning introduces semantic rules that build upon ontology trees of basic actions known by the robot (10). These frameworks are usually limited by the quality of the datasets and the adaptability of the basic actions. Using Semantic Learning, in Beetz et al. (11), a robot application to learn how to cook a pancake using a recipe from wikiHow, an online web with instructions for daily actions, was proposed.

In Deep Learning, one of the concepts that is more interesting to Action Generalization is the concept of latent space. The latent space of a certain input space is known as a set of most relevant/representative features which may be hidden within that input. A latent space can be extracted using Deep Neural Networks. What is interesting about these hidden features is that they allow comparing initially different inputs in this common latent space. This has resulted in the emergence of applications using latent spaces, such as Style Transfer. In Style Transfer, the latent space of different inputs is combined to create new outputs that include a combination of features of these inputs (12).

1.2 The goal of this thesis

To state that the goal of this thesis is to solve the Action Generalization problem in robotics would be way too ambitious for the scope of a doctoral thesis. The goal of this thesis can be summarized in three points:

- Study the state of the art of Action Generalization related methods in robotics.
- Continue the work started by Victores et al. (13) and continued by Morante et al. (3) in the area of Action Generalization.
- Develop new applications with the main goal of improving the Action Generalization capability in robotics using and advancing on state of the art artificial intelligence techniques.

Following these goals, two different approaches are proposed in this thesis. First, a study and improvement of Continuous Goal-Directed Actions (CGDA), initially proposed by Morante et al. (3), is proposed in Part II. In this part, CGDA is improved and adapted to work with dynamic real robot environments. In Part III, Neural Policy Style Transfer (NPST) is proposed as an application of the state of the art of machine learning techniques within an Action Generalization approach. In NPST, Style Transfer is proposed as a way to adapt a base robotic motion to different contexts and situations with the introduction of different Styles selected by the user.

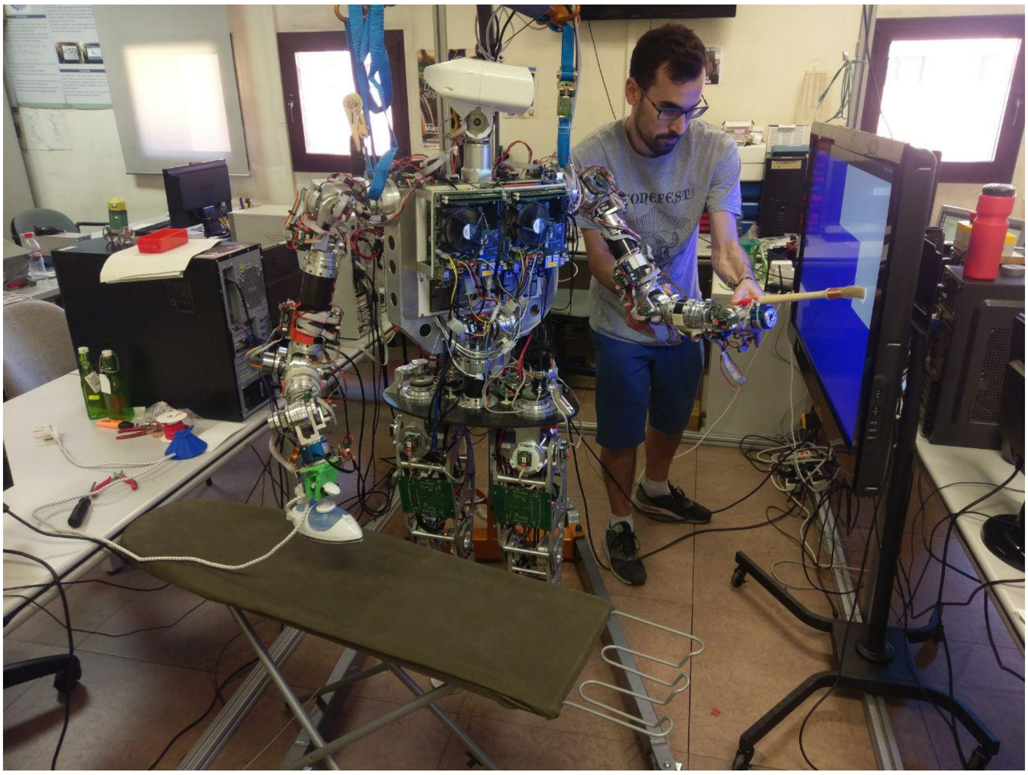

Figure 2: User demonstration of the painting action using the humanoid robot TEO within the CGDA framework.

1.2.1 Improving Continuous Goal-Directed Actions (CGDA)

CGDA is a Learning from Demonstration framework for Action Generalization initially proposed by Morante et al. (3). In CGDA, actions are encoded as *feature trajectories*. These features trajectories are extracted from the effects the actions produce on the environment. Actions are not limited to joint or Cartesian positions as in classical Learning from Demonstration frameworks. A *generalized trajectory* can then be defined using the trajectory extracted from the user, as in Fig. 2. This generalized trajectory can be later executed by the framework.

While encoding the action as a feature trajectory presents many advantages in terms of Action Generalization capabilities, the robot joint trajectories that executes these actions may not be encoded within the model. To solve this, robot joint trajectories are computed using evolutionary algorithms. Due to evolutionary algorithms requiring large numbers of evaluations to converge, these algorithms are performed within a simulated environment. These simulated environments require to previously obtain a model of the environment. Obtaining a model is an expensive operation and, due to the long times that it requires for the evolutionary algorithm to converge, by the end of the optimization process this model is usually outdated. Any changes that may have happened in the environment during the evolutionary algorithm computation increase the probability that the solution obtained is no longer valid and a new one is required. In this thesis, we deal with this problem in two different ways:

- Reducing the number of required evaluations in evolutionary algorithms as a way to reduce computational time and make achievable, in some contexts, the introduction of real world evaluations.
- Proposing the Online Evolutionary Trajectories (OET) algorithm to drastically reduce computational time between real motor executions. This allows regular updates of the generated model to work in real and dynamic environments.

1.2.2 NPST: Introducing Style Transfer in robot actions

Style Transfer is the process of transferring the *Style* of A to B without changing the *Content* of B. The definition of Content and Style is not unique and can vary between different areas of research and between different authors and methods. Style Transfer has been applied to different areas such as computer vision, natural language processing, and animation. The introduction of Deep Neural Networks proposed by Gatys et al. (1) supposed a breakthrough in this area.

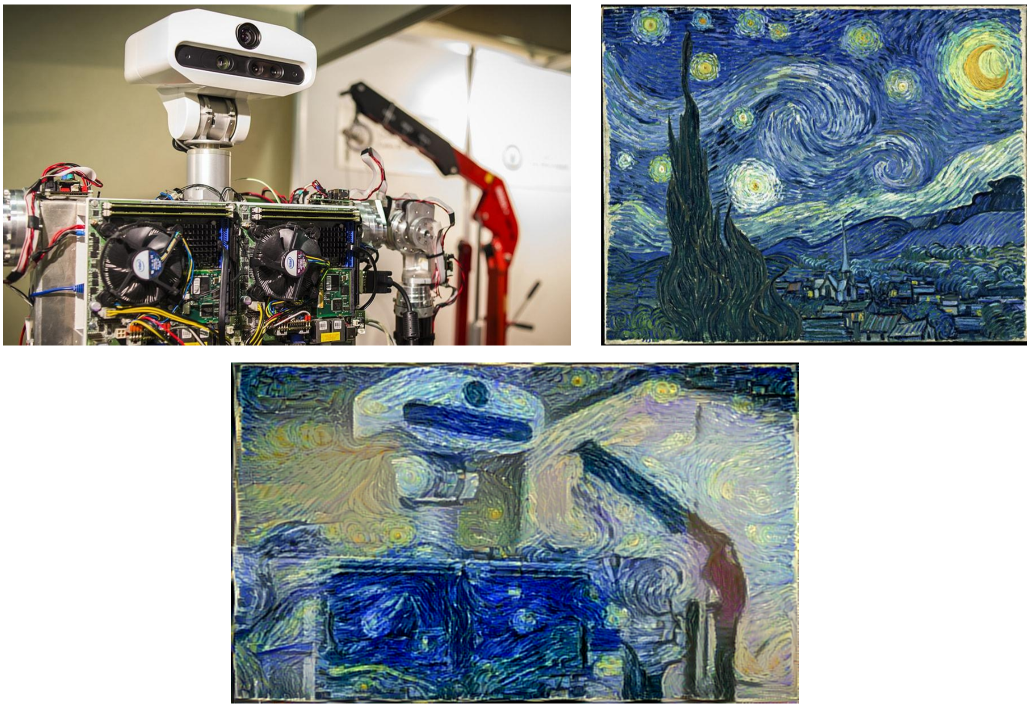

Figure 3: Style Transfer example using the algorithm proposed by Gatys et al. (1) with an image of TEO and the Starry Night of Vincent Van Gogh.

In the work proposed by Gatys, a pre-trained VGG-19 (14) network is introduced to extract latent features of the input images. Using the layer outputs of this network as the latent features of the images, the Style was defined by Gatys as the low level features extracted by the first layers of the network. The Content was defined as the high level features extracted with the last layers of the network. Using an optimization algorithm, a loss function was defined such as the differences between the Content and Style features of the output image were minimized with respect the Content and Style features of the selected Content and Style base images. An example of the results obtained using the algorithm proposed by Gatys et al. is depicted in Fig. 3.

In this thesis, Neural Policy Style Transfer (NPST) is proposed as a way to introduce and transfer Styles between robotic actions. This allows robotic actions to be potentially adapted and modified to different contexts and environments. New algorithms can be proposed where learned actions are adapted to new contexts using Styles defined by actions that have already been successfully applied to that contexts. In NPST, the same learned action is executed after introducing two different moods, fall and nervous. Neural Policy Style Transfer TD3 (NPST3), an advanced version of NPST, is proposed as a way to work with continuous action spaces. Here, the algorithm allows the operator to perform an online teleoperation of the robot while introducing different Styles in the robot motions. The goals presented in this part of the thesis can be summarized in the following two points:

- The study and implementation of state of the art of machine learning techniques in the context of Action Generalization.
- The introduction and adaption of Style Transfer within a robotic framework.

1.3 Relevant concepts and elements

For the definition and study of CGDA and NPST, some additional background is required. The goal of the following sections is to provide an introduction to the base concepts required for the proposed methods. The idea of this section is to serve as a reference point to the reader for the rest of this document.

1.3.1 Elements of Learning from Demonstration

Learning from Demonstration (LfD), also known as Programming by Demonstration (PbD) or Imitation Learning, is a learning paradigm where the goal is to create a framework that allows the robot to learn via examples provided by a demonstrator. The work presented by Billard et al. (15) is a perfect reference for a starting point. The examples or *demonstrations* are defined as expert executions of the learning goal action provided by the user. These demonstrations are typically introduced in the framework by an external recording system such as an RGB-D sensor. Other methods can be used such as kinesthetic learning (16), where the user directly moves the robot for the recording of the expert demonstrations. The *demonstrator* is usually the human operator, but the action demonstrations may come from different sources like video recordings, or even executions from other robots. One of the most important advantages of LfD is giving the ability to non-technical users to teach new actions to the robot.

1.3.2 Elements of Evolutionary Algorithms

Evolutionary algorithms are a subset of global optimization algorithms, inspired by biological evolution (17). These algorithms introduce biological inspired concepts such as mutation, crossover (reproduction) and selection. The *population* of individuals is defined as the group of individual agents that explore, mutate and reproduce over the fitness function. The goal is to find the optimal agent that maximize the obtained fitness defined by the fitness function. The *fitness function* is a mathematical function that maps fitness values with possible agents states. The definition of the fitness function defines the behavior of the optimal agent and allows the encoding of the different action goals. The *evaluation* step is defined as computing the fitness value of the individuals of the population. A complete *iteration* happens when a complete evolutionary step is executed over all the population.

1.3.3 Elements of Reinforcement Learning

Reinforcement learning is a learning framework where an agent is trained to maximize a reward signal (18). The agent behavior in reinforcement learning is defined by the policy. The *policy* is in charge of mapping states with actions. The *reward* signal directly controls the policy behavior. Low reward actions are usually avoided by the policy. The *state* is defined as a unique combination of a set of selected available features. These are the elements that essentially compose a Markov Decision Process. Additionally, the *value function* is defined as the expected reward the agent can expect to obtain in the long run as a function of the current state and a given policy. Finally, the *model* is an optional element that encodes the behavior of the environment. Models can be used, for example, to predict future states and rewards.

Deep Reinforcement Learning is a set of developments that propose the introduction of Deep Learning architectures as function approximators within Reinforcement Learning. For example, Mnih et al. (2) proposed the introduction of a deep neural network to define the value function of their Reinforcement Learning framework.

CONCLUSIONS OF PART I: INTRODUCTION

In this part of the thesis, an introduction to the problem of Action Generalization in robotics has been proposed. This problem focuses on giving robots the ability to adapt actions to different contexts and environments. Based on this idea, the goals presented in this thesis to improve the Action Generalization capabilities of robots were presented. A brief introduction to the two main proposed approaches to achieve these goals was introduced. To finish, the most relevant concepts that appear in this work have been defined.

This part of the thesis is intended to serve as a reference point to the reader. Relevant concepts and ideas that later appear in this thesis are defined for reference. The rest of this document is organized to represent the continuous research project that this thesis has been. A study of the state of the art of the existing relevant topics is presented within each part.

The problem of Action Generalization is one of the hardest problems in robotics. In this thesis, two different approaches are proposed as a way to improve the Action Generalization capabilities of robots. The first approach focus on improving CGDA, while the second approach introduces Style Transfer within a Deep Reinforcement Learning architecture.

Part II

Continuous Goal-Directed Actions

2. CONTINUOUS GOAL-DIRECTED ACTIONS

Continuous Goal-Directed Actions (CGDA) is a Learning from Demonstration framework for the encoding of actions as environment feature trajectories. In Learning from Demonstration, the learning framework of the robot is given expert demonstrations of the desired action by the user. These demonstrations can come from action executions performed by the user, moving the robot (16), recorded videos or even other robot executions. The demonstrations are then used by the framework to extract a generalized trajectory to learn the demonstrated action. One critical aspect that defines the Learning from Demonstration framework is the selection of the model that encodes this generalized trajectory. In Calinon and Billard (19) and Calinon et al. (9), Hidden Markov Models and Gaussian Mixture Models are used to extract generalized robot Cartesian and joint space trajectories. In Dynamic Motion Primitives (DMP) (20), these trajectories are discretized using a database of low level predefined control laws that define a set of low level Cartesian space trajectories. Finally, in CGDA, actions are encoded as features trajectories that encode the effects the demonstrated action produced on the environment (3).

One of the most important advantages of CGDA is that encoded generalized trajectories are not limited by the encoding of Cartesian and robot joint positions. A generalized trajectory in CGDA can be encoded, for example, using only the percentage of painted area each time step. This solves the correspondence problem (21) in all situations where a certain kinematic configuration is no explicitly required to describe the action.

The selection of the encoded scalar features for every action can be handcrafted, or can be selected using the demonstration and feature selection algorithms proposed by Morante et al. (22). Here, the authors combined the introduction of Dynamic Time Warping (DTW) (23) and z-score to extract the consistency of features between demonstrations. The most consistent features are selected as relevant and tracked by the CGDA system. Features with low consistency between demonstrations are discarded.

Since robot joint trajectories may not be explicitly encoded in CGDA, these trajectories have to be computed inside the framework. As a consequence of this, evolutionary algorithms are introduced as a way to compute these trajectories. These algorithms require a large number of evaluations in the target environment before generating the robot joint execution trajectory. Performing all these evaluations on the actual physical robot is extremely costly and most of the times infeasible. As a solution, these evaluations are performed in simulation using models of the environment. The optimal trajectory obtained in the simulation is then executed on the real robot.

2.1 The CGDA framework

The CGDA framework is divided in three stages: Generalization, Recognition and Execution. In CGDA, an action is encoded as a trajectory in a feature space of m scalar features. These features encode the changes the demonstrated action produced on the environment and may include visual features, forces exerted by a given actuator, Cartesian positions of moving objects in the environment, or even joint trajectories. The execution of the encoded actions is achieved as an optimization problem with the goal to match the generalized feature trajectory encoded for the action with the feature trajectory generated by the CGDA execution step. This optimization problem is computed using evolutionary algorithms. The result is a robot joint trajectory that can be directly executed by the robot.

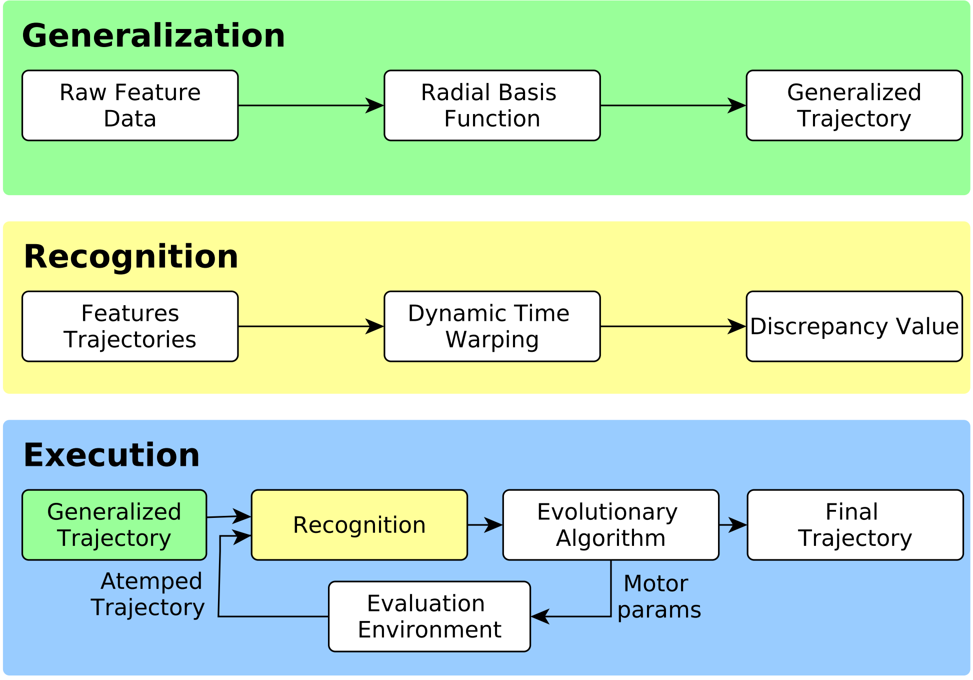
Generalization
Raw Feature Data
Radial Basis Function
Generalized Trajectory
Recognition
Features Trajectories
Dynamic Time Warping
Discrepancy Value
Execution
Generalized Trajectory
Recognition
Evolutionary Algorithm
Final Trajectory
Atemped Trajectory
Evaluation Environment
Motor params

Figure 4: Continuous Goal-Directed Actions (CGDA) framework diagram.

A conceptual scheme of the CGDA framework is depicted in Fig. 4. In the Generalization stage, the generalized feature trajectory is extracted from the user demonstrations using the raw feature data extracted from the sensors. The Recognition step fulfills a double role: it can be used to compare the similarity between two actions given their feature trajectories, or it can be used as an evaluation stage where the performance of the executed action with respect the generalized action is evaluated. At the end of this stage, a discrepancy value between the input trajectories is obtained. In the Execution stage, the generalized trajectory and the discrepancy value obtained in the previous stages are introduced in an evolutionary algorithm to compute the optimal robot joint trajectory. This final robot joint trajectory is then executed by the robot.

2.2 Generalization

In the Generalization stage, a generalized m-dimensional feature trajectory X is extracted from user demonstrations, where m is the number of tracked scalar features. This generalized feature trajectory encodes the action defined by the user. In order to extract X, first, all user demonstrations are normalized in time. The generalized trajectory X is defined as a normalized discrete time trajectory with n equitemporal intermediate goals. The number of intermediate goals is computed using Eq. 2.1.

$$n = \lfloor \frac{D_{time}}{T_{min}} \rfloor \tag{2.1}$$

Where D_{time} is the average duration of the user demonstrations before the normalization step, and T_{min} is the desired minimum time interval between intermediate goals. Given a fixed T_{min}, longer actions will be assigned with a larger number of intermediate goals to avoid losses of information. The generalized trajectory X is then defined following Eq. 2.2.

$$X = (X_1 \cdots X_j \cdots X_n) = \begin{pmatrix} x_{00} & \cdots & x_{0n} \\ \vdots & \vdots & \vdots \\ x_{m0} & \cdots & x_{mn} \end{pmatrix} \tag{2.2}$$

Where X_j is an intermediate goal of the generalized trajectory, and x_{mn} is the value of the m scalar feature at the time corresponding to the n intermediate goal.

Given d as a set of user demonstrations, d_i can then be defined as the value of the scalar feature i over this set of user demonstrations. The value of any intermediate goal x_{ij} can be computed following Eq. 2.3.

$$x_{ij} = \frac{1}{|d_i \in [j, j+1]|} \sum_{d_i \in [j,j+1]} d_i \tag{2.3}$$

Using these intermediate goals a generalized feature trajectory can be computed using a Radial Basis Function interpolation (24) as in Eq. 2.4.

$$f(x) = \sum_{i=1}^{m} w_i \phi(\|x - x_i\|) \tag{2.4}$$

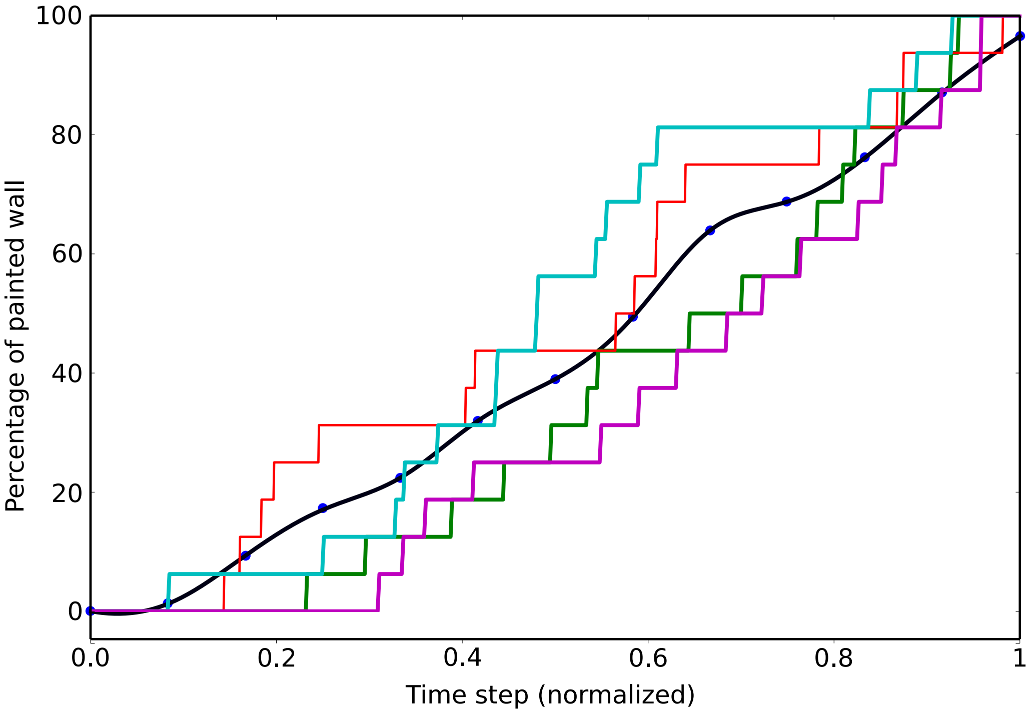

Figure 5: Plot of a 1-dimensional feature action generalization. Colorized lines depict different demonstrations. The black line is the generalized feature trajectory obtained from all the demonstrations.

In Fig. 5 there is an example of a generalized trajectory obtained using a set of user demonstrations. This feature trajectory encodes the “paint” action as defined by Morante et al. (3). The goal of this action is to use a brush attached to the robot end effector to paint a wall.

2.3 Recognition

In the recognition stage, generalized feature trajectories are used as reference inputs to compare and measure the performance of executed actions or to identify new ones. In this stage, the discrepancy between the observed feature trajectory O defined by the executed or new action and the generalized one X is obtained. This discrepancy depicts how different these two trajectories are. Executed trajectories with high discrepancy are associated with a poor execution performance. New trajectories with low discrepancy are identified as similar actions that, in some scenarios, could be added to the demonstration database.

Dynamic Time Warping (DTW) is introduced as an optimization method for temporal alignment (23). One of the advantages of DTW is that it allows comparing feature trajectories from actions executed with different velocities. In CGDA, an intermediate result obtained with DTW is used as the discrepancy value. In DTW, all paired combinations of points between the features trajectories are compared using the L^2 norm. Then the Cost Matrix (CM) with all these values is obtained as presented in Eq. 2.5.

$$CM = \begin{pmatrix} d(o_0, x_0) & \cdots & d(o_{n'}, x_0) \\ \vdots & \vdots & \vdots \\ d(o_0, x_n) & \cdots & d(o_{n'}, x_n) \end{pmatrix} \tag{2.5}$$

Using this cost matrix, an optimal cost path can be obtained such as the pairing combinations selected minimize the total cost value of the two trajectories. This path is defined as the lowest cost path. The total cost value $C_P(X,Y)$ of a given path P is computed as the sum of all the local costs C of each alignment as depicted in Eq. 2.6.

$$C_P(X,Y) = \sum_{l=1}^{L} C(o_{n'l}, x_{nl}) \tag{2.6}$$

Where L is the length of the path P.

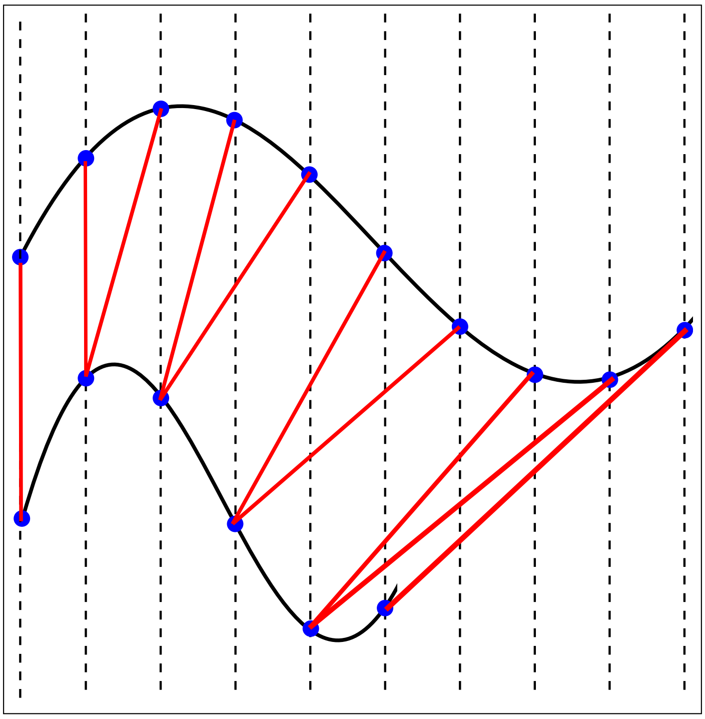

Figure 6: Case example of applying DTW between two trajectories. The two trajectories used in this example are the same, but with different time executions.

The result of applying DTW in a 2-dimensional feature space is the temporal alignment path C_P that minimizes the total cost taking in account both dimensions. An example of the result of applying DTW to two 1-dimensional trajectories is depicted in Fig. 6. In the case of m-dimensional trajectories, the total cost D is defined as the sum of the costs obtained for the optimal alignment of each feature dimension as in Eq. 2.7.

$$D = \sum_{i=1}^{m} C_{P_{norm}}(O_i, X_i) \tag{2.7}$$

This value D is defined as the discrepancy value used inside the CGDA framework.

2.4 Execution

In CGDA, joint motor parameters may not be explicitly encoded within the framework. Additionally, conventional methods for the execution of the trajectory, such as Inverse Kinematics, are only useful in situations when the feature trajectory encoding the action only encodes Cartesian positions of the end effector. In CGDA, this is not the general case. Feature trajectories usually encode different features like the Cartesian positions of objects, visual features, force sensor information... In these situations, the robot joint motor trajectory corresponding to the action has to be obtained as an optimization problem. The goal is to find the joint parameters that reduce the discrepancy between executed feature trajectories and generalized ones. For this optimization problem, evolutionary algorithms are introduced within the CGDA framework. Evolutionary algorithms require performing large numbers of evaluations to converge.

In order to increase the performance of evolutionary algorithms within the CGDA framework, three different Evolutionary Strategies were proposed at Morante et al. (3): Full Trajectory Evolution (FTE), Individual Evolution (IE), and Incrementally Evolved Trajectories (IET). The following sections introduce these strategies as defined in previous literature.

2.4.1 FTE: Full Trajectory Evolution

The first algorithm proposed is FTE. The pseudocode of this algorithm is depicted in Algorithm 1. In FTE, each individual encodes a full robot joint trajectory U. The obtained solution U defines the required full robot joint trajectory to reach all the defined intermediate goals.

The *initialize* and *evolve* steps are defined by the evolutionary algorithm selected. The termination conditions can be defined by the user. The total number of parameters for each individual is $DoF \cdot n$, where DoF is the

Algorithm 1 Full Trajectory Evolution (FTE)

```
procedure FTE($X$)
    individuals ←initialize
    while not termination_conditions do
        for each individual do
            $U$ ←evolve($DoF \cdot n$)
            $O$ ←mental_execution($U$)
            $f$ ←mental_recognition($O, X$)
        end for
    end while
    motor_execution($U$)
end procedure
```

number of Degrees of Freedom and n is the number of intermediate goals. In the *mental_execution* step, each individual is executed within a simulated environment. The feature trajectory generated with this execution is then introduced to the *mental_recognition* step. This step measure the discrepancy value obtained by the solution. This discrepancy value is assigned to the individual as the individual fitness value f used in the *evolve* step. This strategy tends to be the most computational expensive of the three presented due to the usually large search spaces.

2.4.2 IE: Individual Evolution

The second algorithm proposed was IE as depicted in Algorithm 2. In IE, each intermediate goal defined in the generalized feature trajectory is treated as an independent optimization problem. Individuals are now defined using only *DoF* parameters corresponding to the robot joint positions required to reach an intermediate goal.

Although computationally less restrictive than FTE, this strategy is not suitable for actions where there is a dependency between intermediate goals. This is the case of the “paint” action later presented in this thesis. In CGDA, this action is encoded using the percentage of painted wall. For the execution of this action, each intermediate goal, this percentage

Algorithm 2 Individual Evolution (IE)

```
procedure IE($X$)
    $individuals$ ←initialize
    for $j < n$ do
        while not $termination\_conditions$ do
            for each $individual$ do
                $U_j$ ←evolve($DoF$)
                $O_j$ ←mental_execution($U_j$)
                $f$ ←mental_recognition($O_j, X_j$)
            end for
        end while
    end for
    motor_execution($U$)
end procedure
```

may has to be increased a certain value. In IE, when performing the *mental_execution* and *mental_recognition* step, the framework does not have information about the effects of the execution of previous intermediate goals. Although some intermediate goals may individually increase the percentage of painted wall as expected, this may not happen when executing the full generated robot joint trajectory. Some intermediate goals may result in painting already painted areas by previous intermediate goals executions. The result is an incompletely painted wall with some areas painted more than once.

2.4.3 IET: Incrementally Evolved Trajectories

In IET, Algorithm 3, joint positions U_j are evolved for each intermediate goal after the mental execution of $U_{[0,j-1]}$. IET works in a similar way than IE, but with one critical difference. In IET, before the computation of each intermediate goal, all the previous joint positions are executed in the simulated environment. This updates the environment and solves the problems presented in IE. In the case of the "paint" action, already painted areas are presented painted for each intermediate step. Painting these areas will not increase the percentage of painted wall and new solutions will have

Algorithm 3 Incrementally Evolved Trajectories (IET)

```
procedure IET($X$)
    $individuals$ ←initialize
    for $j < n$ do
        while not $termination\_conditions$ do
            for each $individual$ do
                mental_execution($U_{[0,j-1]}$)
                $U_j$ ←evolve($DoF$)
                $O_j$ ←mental_execution($U_j$)
                $f$ ←mental_recognition($O_j, X_j$)
            end for
        end while
    end for
    motor_execution($U$)
end procedure
```

to be found. In the experiments performed in the original paper, this was presented as the most successful strategy of the three and the one chosen to be implemented inside the CGDA framework. The main advantage of IET is to combine the reduced search space of IE with the high performance of FTE.

2.5 Conclusions

In this chapter, the CGDA framework for action generalization has been introduced. CGDA is a Learning from Demonstration framework where actions are encoded as feature trajectories. These features trajectories are defined as the effects these actions produce on the environment. The CGDA framework is composed by three stages: Generalization, Recognition, and Execution. In the Generalization stage, the generalized feature trajectory that encodes the action is generated. In the Recognition stage, a discrepancy value is obtained between two different feature trajectories using DTW. This discrepancy value can be used to evaluate the performance of an executed action or to recognize new ones. Finally, in the execution stage, evolutionary algorithms are introduced as a way to generate the corresponding robot joint trajectories for the robot execution.

Since evolutionary algorithms methods are usually computationally expensive, different evolutionary strategies were proposed with the original framework. These strategies are aimed at improving the performance and efficiency of the evolutionary algorithms introduced. The execution of any of these strategies still requires the evaluation of hundreds or thousands of solutions in a simulated environment. Even for the reduced search space defined by the IET strategy, this operation is still computational expensive and time consuming. For some actions, the resulting trajectory may be outdated at the time to be executed in the robot due to changes produced in the environment while being generated. As a solution, in the first part of this thesis, a study of how to improve the performance of CGDA with dynamic real world environments is presented.

3. REDUCING EVALUATIONS IN CGDA

The CGDA framework requires the use of evolutionary algorithms for the generation of robot joint trajectories. These algorithms converge to the optimal solution by using a large number of evaluations. These evaluations are performed in computationally expensive simulated environments that require precomputed environment models. Due to the dynamic and complex nature of real world environments, these models are usually imprecise and costly to develop. In this chapter, a study of how to reduce the number of evaluations in evolutionary algorithms is introduced. The goal is to reduce the computational cost of performing evolutionary algorithms steps, and, in some situations, potentially allow the framework to perform the evaluations directly in the real world. Two different techniques to reduce the number of evaluations will be studied and presented. The first one consists on a set of different methods that reduce the number of evaluations by changing the base of the evolutionary algorithms definition. The second one introduces constraints in evolutionary algorithms to guide the convergence of the algorithm to achieve faster solutions. Both of these alternatives will be studied and implemented within the CGDA framework. The results will be compared with respect a common baseline using Steady State Tournament (SST) as proposed in the first implementations of CGDA. SST is part of the sub-group of evolutionary algorithms called genetic algorithms (17).

3.1 Introducing Approximations

The problem of reducing the number of evaluations in evolutionary algorithms is not exclusive of CGDA. Real world scenarios or time sensitive applications are also affected by the large number of evaluations required for evolutionary algorithms to converge. Different methods for the introduction of approximations within evolutionary algorithms have already been proposed. These methods aim at introducing approximations to reduce the problem complexity and with it the number of evaluations required for converging. They can be classified in three groups as a function of where the approximation is introduced (25):

- **Problem Approximation:** In this category, the problem is simplified by introducing approximations over the original problem definition. These methods require a deep study of the problem to introduce these approximations without changing the original goal.
- **Functional Approximation:** In these methods, the approximations are introduced in the fitness function defined by the problem. The fitness function is simplified with an approximate similar function that is simpler to solve and less computational expensive. In Vincenzi and Savoia (26), a speed up of a Differential Evolution algorithm is achieved by introducing second order approximations in the fitness function. The speed up of the system also came with a reduction in the number of evaluations required by the system to converge.
- **Evolutionary Approximation:** In Evolutionary Approximation, the base definition of the evolutionary algorithm is modified. There are two main groups of Evolutionary Approximation algorithms: Fitness Inheritance (FI) (27) and Fitness Approximation (FA) (25).

  In FI, only a proportion of the population is evaluated. The rest of the population fitness is computed through an approximation fitness function. This approximation fitness function does not require the evaluation of the approximated individuals. The fitness value of these individuals is computed as a function of the fitness value of the evaluated individuals. The number of required evaluations is reduced by the proportion of the population computed through this approximate fitness function. Barbour et al. (28) proposed the introduction of FI methods for the optimization of chemotherapy dose schedules. The design of chemotherapy dose schedules tends to be a complex problem with many variables (effects, schedules, drugs combination...) that requires large number of evaluations to converge. This is usually a computationally expensive problem and, at the same time, a critical scenario where saving computational time is crucial.

In FA algorithms, the population is divided in clusters over the fitness function. Each cluster has a fitness value assigned. Every time a new particle is generated or moved to a new cluster, the particle fitness is set to the one of that cluster. In Esparcia-Alcázar and Moravec (29), FA is implemented within the computer game Unreal Tournament 2004$^{\text{TM}}$ for bot learning. In this scenario, evaluations must be performed in real playtime, making them extremely constrained in terms of available time. Another example is Bertini et al. (30). Here, FA is introduced to reduce the number of evaluations required for the optimization of a start-up phase of a combined cycle power plant. This is a complex problem that requires a complex computational cost model of the environment.

3.2 Particle Swarm Optimization in CGDA

For the original proposition of CGDA, an SST algorithm was chosen as the evolutionary algorithm for the generation of joint robot trajectories. The most advanced methods for introducing approximations within evolutionary algorithms are proposed using Particle Swarm Optimization (PSO). For the introduction of these approximations within the CGDA architecture, an implementation of PSO is proposed in this thesis. The results obtained will be later compared to the ones obtained with SST as in the original framework.

3.2.1 Particle Swarm Optimization

The Particle Swarm Optimization (PSO) algorithm was initially proposed by Kennedy and Eberhart (31). PSO was designed to be a method based on introducing social interactions behaviors in the population of individuals. In PSO, a population of particles is randomly initialized within the fitness function. Then, each movement step, the positions of the particles are updated using a velocity v_i computed as in Eq. 3.1. This velocity is a function of the particle's current velocity, the position of the best particles in the population, and the best position in the particle history (32). Each time the population has moved, an evaluation step is performed and the fitness value of each particle updated.

$$\begin{cases} x_i(t+1) = x_i(t) + v_i \\ v_i(t+1) = wv_i(t) + R_1c_1(P_i - x_i(t)) + R_2c_2(P_g - x_i(t)) \end{cases} \tag{3.1}$$

Where w is the inertia weight, R_1 and R_2 are two random values, and c_1 and c_2 are constants. P_i is the best position in the particle history, and P_g is the position of the best particle in the swarm.

3.2.2 Adaptive Fuzzy Fitness Granulation PSO

Davarynejad et al. (33) initially proposed the idea of Adaptive Fuzzy Fitness Granulation (AFFG) as an evolutionary optimization method. This method can be considered as part of the FA algorithms, where individuals are clustered inside granules of individuals. These granules are defined within the fitness function using Gaussian distributions. New individuals are assigned to granules similar to them. The fitness value of new individuals is set to the one corresponding to the assigned granule. No evaluations are required for these individuals. If a new individual does not fit any already existing granule, the new individual is evaluated and a new granule is created. The fitness value of this new granule is set to the fitness value obtained when evaluating the new individual. The results depicted on the experiments performed by Akbarzadeh-T et al. (34) show a reduction, in some scenarios, in the number of evaluations of 90% while reaching a similar performance. In this thesis, an implementation of AFFG with PSO (AFFG-PSO) is proposed as an original contribution.

3.2.3 Fitness Inheritance PSO

Fitness Inheritance (FI) was introduced by Smith et al. (27) as a way to reduce the high computational cost of evolutionary algorithms. The initial idea was to only perform the evaluation of a portion of the population. The rest of the population fitness value was approximated as a function of the fitness value of the already evaluated individuals.

In the experiments performed in Ducheyne et al. (35), the goal was to measure the feasibility of performing FI with real world scenarios. The results showed how FI algorithms were able to converge when facing problems with convex fitness functions. This was not the same when facing non-convex problems. In these situations, the algorithms were not able to find the optimal solution. In Reyes-Sierra and Coello (36), the authors proposed an implementation of FI within a PSO framework (FI-PSO). The results of this paper depicted the convergence of the algorithm also for the non-convex problems. Later, in Reyes-Sierra and Coello (37), the authors proposed a set of methods to improve the performance of the initially proposed FI-PSO method. One of these methods obtained the best results by introducing the approximation fitness formula depicted in Eq. 3.2.

$$f_i(t+1) = f_i(t) + \frac{c_1 R_1 P_i(t) + c_2 R_2 P_g(t)}{1 + c_1 R_1 + c_2 R_2} \tag{3.2}$$

3.3 Introducing Constraints

Within the scope of genetic algorithms, some literature related to the introduction of constraints in the search space of the evolutionary algorithm exists. For accurate model generation, constraints are studied as a way to represent infeasible positions within real world scenarios. Genetic algorithms are not designed to work with constrained problems. Different strategies to deal with the introduction of constraints within genetic algorithms have already been proposed. These strategies can be divided in different groups classified by the way they deal with solutions defined within constrained states (38). In this thesis, constraints are introduced to reduce the search space of the robot and, as a consequence, the number of required evaluations to converge. The most relevant strategies are:

- **Rejecting strategy:** This strategy simply rejects solutions that correspond to constrained states. These constrained solutions are not taken in account by the algorithm at the time of computing the fitness. Only non constrained solutions are evaluated. This strategy fails if all the obtained solutions in one iteration are rejected. An example of this strategy is the work proposed by Zhao and Sannomiya (39). Here, the authors introduced a rejecting strategy for the problem of designing a flow shop schedule. In flow shop schedule problems, a big part of the action combinations are infeasible ones.

- **Repairing strategy:** This strategy focus on maximizing the number of solutions within the non constrained space each iteration. Constrained solutions, instead of being discarded, are "repaired" and transformed to non constrained ones. This repairing step can sometimes be as complex and computational expensive as the evolutionary algorithm itself. Chootinan and Chen (40) proposed the introduction of a gradient based function within the constrained area. This gradient function is used to redirect constrained solutions to non constrained states.
- **Modifying genetic operator strategy:** In this strategy, the genetic algorithm hyper-parameters are modified to only allow the generation of solutions within non constrained states. This is the case for example of defining the individuals using constrained values.
- **Penalty strategy:** This strategy is equivalent to a rejecting strategy but with constrained solutions being assigned a penalty fitness value instead of being directly rejected. Constrained areas are converted to penalized ones. Constrained solutions are equivalent to non constrained ones with penalization values. The introduction of this penalization value can be done in two ways:

$$eval(x) = \begin{cases} f(x), & if \quad x \in F \\ f(x) + p(x) & otherwise \end{cases} \tag{3.3}$$

$$eval(x) = \begin{cases} f(x), & if \quad x \in F \\ f(x)p(x) & otherwise \end{cases} \tag{3.4}$$

Where $f(x)$ is the fitness function of the problem, $p(x)$ the penalty function, and F the non constrained space.

From all of these strategies, the penalty strategy is the one with the most related literature. Penalty strategies algorithms can be classified according to how the penalty is obtained (41) in the following groups:

- **Death Penalties:** The fitness penalty value for all the solutions inside the constrained space is fixed to ∞.
- **Static Penalties:** All the solutions in the constrained space are assigned with the same constant fitness value.
- **Dynamic Penalties:** The current number of iterations performed over the population is used to compute the fitness penalty value.
- **Adaptive Penalties:** The fitness penalty value is defined as a function of the population current state. In Ben Hadj-Alouane and Bean (42), if all the solutions obtained in the last iteration were located in the constrained space, the penalty value was increased. On the contrary, if none of the solutions were part of the constrained space, the penalty value was decreased.
- **Annealing Penalties:** Annealing Algorithms are introduced to compute the fitness value as function of the temperature value (43).

Several algorithms based on these strategies have been proposed by different authors (44, 45, 46, 47). Some of these algorithms were proposed as a combination of different strategies. Chang (48) proposed a system using a combination of death and static penalties to control the water levels of a water reservoir. Death penalties are used to penalize solutions outside the water reservoir boundaries. A static penalty is introduced to penalize solutions that create water levels outside the safe levels of the system.

3.4 Approximations and Constraints in CGDA

Evolutionary algorithms approximations and constrained genetic algorithms are implemented and studied within the CGDA framework in order to reduce the number of evaluations required.

In terms of introducing evolutionary algorithms approximations, two different algorithms are proposed and implemented in this thesis: AFFG-PSO as an original contribution of this thesis; and FI-PSO as proposed by Reyes-Sierra and Coello (37). To compare the performance of these methods, two base algorithms are also implemented as the baselines: Steady State Tournament (SST) (49) as in the original CGDA framework; and Particle Swarm Optimization (PSO) (31) as the vanilla algorithm.

Algorithm 4 AFFG-PSO

```
procedure AFFG-PSO($similarity\_threshold, max\_num\_granules$)
    $individuals, granules \leftarrow$ initialize
    while not $termination\_conditions$ do
        $Pbest \leftarrow$ find_best_particle($individuals$)
        for each $individual$ do
            $individual\_pos \leftarrow$ compute_velocity($individual, Pbest$)
            $max\_similarity \leftarrow$ compute_max_similarity($individual, granules$)
            if $max\_similarity > similarity\_threshold$ then
                $individual\_fit \leftarrow granule\_fit$
            else
                $individual\_fit \leftarrow$ evaluate($individual$)
                $granules \leftarrow$ generate_new_granule(individual)
            end if
        end for
        $granules \leftarrow$ update_granules($granules, max\_num\_granules$)
    end while
end procedure
```

The pseudo code of the proposed AFFG-PSO algorithm is depicted in Algorithm 4. Here, *find_best_particle* and *compute_velocity* are defined as in the vanilla PSO algorithm. The *compute_max_similarity* function computes the similarity of the given *individual* with the most similar existing granule. This similarity is computed as proposed by Akbarzadeh-T et al. (34). The *similarity_threshold* is a constant value that defines the minimum similarity value required to consider a particle part of a granule. The *update_granules* operation checks the list of existing granules. Less relevant granules are deleted when reached the maximum number of granules defined by *max_num_granules*. Granules are generated following the AFFG algorithm and the population of individuals is evolved as in the vanilla PSO.

In terms of constrained genetic algorithms, two different types of constraints are defined for the experiments of this thesis. These are the spatial and velocity constraints. The spatial constraint scenario introduces a constraint in the Cartesian search space of the robot. This constraint space is defined as the minimum bounding box around the solution space with a dilatation value. Solutions that generate robot end effector positions outside this valid space are treated as constrained ones. The velocity constraint is introduced in the joint velocity space of the robot. When a solution is generated, the joint velocity resulting of that solution is obtained. If the result value is higher than a given threshold, the solution is treated as constrained.

3.5 Experiments

Experiments were performed to test the effects of introducing constraints and approximations within the CGDA framework. The reduction in the number of evaluations was treated as the main parameter to measure. Two sets of experiments were performed. The first set focus on the introduction of evolutionary algorithms approximations in the CGDA framework. This set of experiments introduce the implementation of the four algorithms proposed earlier in this chapter: SST, PSO, AFFG-PSO, and FI-PSO. The second set of experiments introduces the spatial and velocity constraints as defined in the previous section. Here, SST was introduced as the evolutionary algorithm for this set of experiments.

The experiments were performed within a simulated environment using OpenRAVE (50) as the simulation platform. YARP (51) was introduced for the internal communications. The humanoid robot TEO[1] (52) from Universidad Carlos III de Madrid was chosen as the robotic platform. Three joints of the right arm of the robot (two from the shoulder and one from the elbow) were used for the control and generation of the joint robot trajectories. The rest of the joints were fixed to a constant position. The "wax" (defined as "clean" in previous literature (3)) and the already defined "paint" actions were the actions selected for the experiments. One generalized feature trajectory was generated for each of the actions. These actions were chosen in order to have a common baseline with respect previous CGDA literature. For the execution of both actions, only the movement of the right arm of the robot was required. The IET evolutionary strategy was selected as the default evolutionary strategy for all of the experiments. The code required to run the experiments was open-sourced and made available online[2].

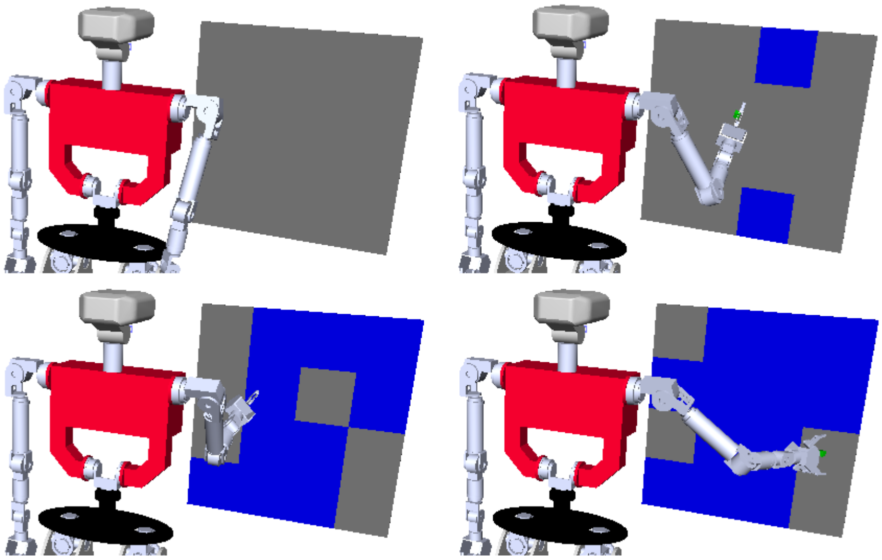

Figure 7: Simulated execution of the "paint" action using the humanoid robot TEO and Openrave.

---

[1] https://github.com/roboticslab-uc3m/teo-main

[2] Source code available at https://github.com/roboticslab-uc3m/xgnitive

In the “wax” action, the goal is to move an object performing a circular trajectory with a diameter of 30 cm for one revolution. This movement is similar to the famous “wax” movement depicted on the film “The Karate Kid” (53). Three scalar features corresponding to the scalar Cartesian position (X,Y,Z) of the object’s centroid are used to encode this action. The generalized trajectory is a circular trajectory of 30 cm of diameter. In the “paint” action, the goal is to paint a wall using a brush connected to the robot end-effector. Only one scalar feature, the percentage of painted wall, is required to encode this action. The generalized feature trajectory is defined as a straight line going from 0% to 100% of painted wall. For simplification, a 4x4 model of a discretized wall is introduced for both the simulation and real world executions. An area of the wall is considered painted when the painting tool of the robot is closer than a defined painting threshold with respect the given area. In real world executions, a monitor in front of the robot is used to depict the simulated wall. Here, the RGB-D sensor of the robot is used to find the position of the painting tool and check if some area of the wall has been painted. Fig. 7 depicts an execution of the “paint” action in simulation.

For the experimental setup, the joint position value for each of the joints was limited to a minimum of -15 and a maximum of 100 degrees. The mutation probability for the SST algorithm was fixed to 60%. The tournament size was set to 3. For the “wax” action, the total number of individuals was set to 50. The termination condition was set to reach a maximum number of iterations without improvement of 3. For the “paint” action, the total number of individuals was set to 10. The termination conditions were set to reach a maximum number of iterations without improvement of 10 or obtaining a zero discrepancy error with the generated solution.

In the Evolutionary Approximation experiments, the PSO inertia weight value was fixed to 1.2 and the maximum particle velocity was limited to a value of 5. The maximum number of granules, for the AFFG-PSO algorithm, was fixed to 3. A proportion of 55% of the particles were not evaluated, in the case of FI-PSO.

In order to measure the performance of each of the selected algorithms, two different parameters were introduced. These two parameters are the number of required evaluations and the discrepancy value. The discrepancy value is measured as a way to study how the reduction of evaluations impacts the quality of the obtained solutions. In addition to this, for the "paint" action, the percentage of painted wall was also introduced as a complementary parameter to this discrepancy value.

3.6 Evolutionary Approximation Results

In the first set of the experiments, the four selected algorithms for Evolutionary Approximation were tested: SST, PSO, AFFG-PSO and FI-PSO. These algorithms were introduced in the execution stage of the CGDA framework. Experiments were performed over the two different actions: "wax" and "paint". The results depicted in this section compare the effects of introducing each of these algorithms.

3.6.1 Wax Results

For each implementation of the four proposed algorithms, 50 repetitions of the execution of the "wax" action were performed.

Table 3: "Wax" Evolutionary Approximation results: average results after 50 repetitions of the "wax" action for each of the proposed algorithms.

Algorithm	Evaluations	Discrepancy
SST	9679	274
PSO	8470	213
AFFG-PSO	5314	434
FI-PSO	3432	362

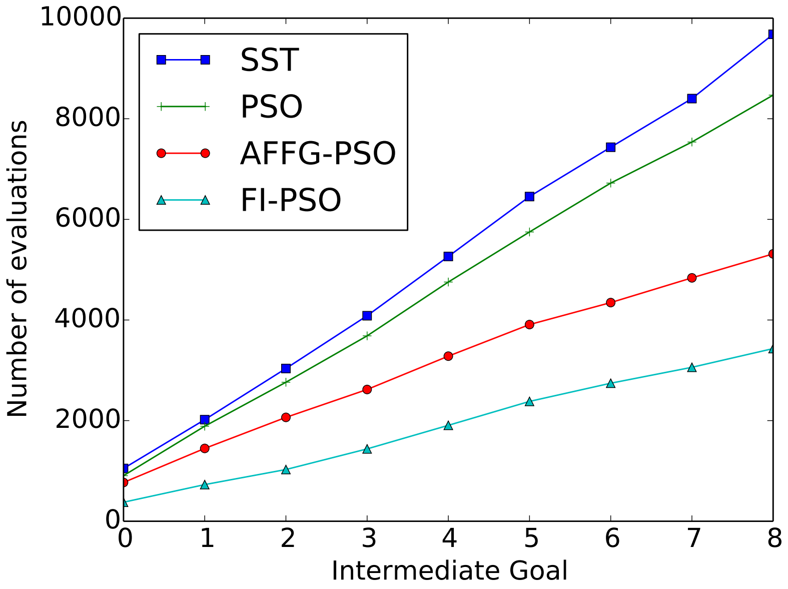

Figure 8: "Wax" Evolutionary Approximation results: total number of required evaluations for the "wax" action execution as a function of the intermediate goals defined in the feature trajectory.

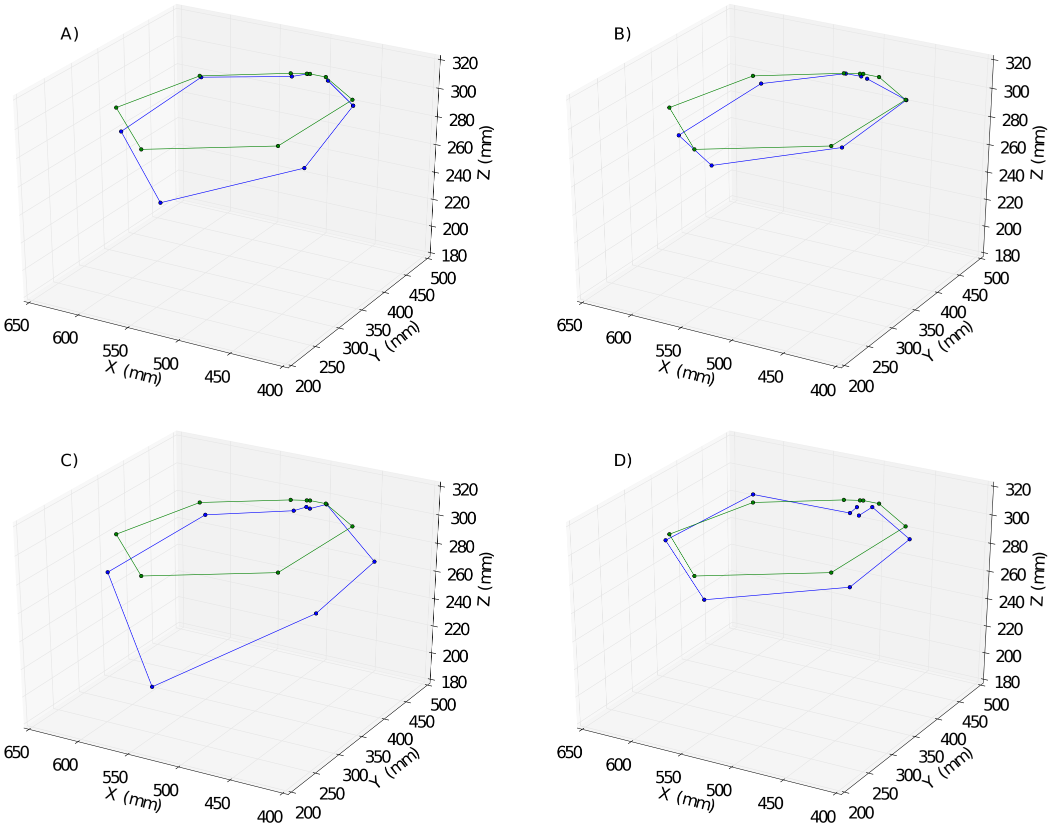

Figure 9: "Wax" Evolutionary Approximation results: feature trajectories obtained with the following algorithms: A) SST, B) PSO, C) AFFG-PSO and D) FI-PSO. The green trajectory depicts the generalized feature trajectory encoding the action. The blue trajectory is the feature trajectory obtained with the execution stage of CGDA.

The results obtained during these experiments are depicted in Table 3. FI-PSO was the method with a lower number of required evaluations. The number of required evaluations was reduced by 65% with an increase of 32% in the discrepancy value with respect the original SST algorithm. In the case of the AFFG-PSO algorithm, this reduction was 45% with an increase in the discrepancy value of 58%. Finally, the vanilla PSO algorithm required 12% less evaluations with a discrepancy value reduction of 22%.

Fig. 8 depicts the total number of performed evaluations as a function of the intermediate goal. In Fig. 9, a plot of the generated Cartesian trajectories for each of the algorithms is depicted. For this action, this Cartesian trajectory correspond to the feature trajectory used to encode the "wax" action.

3.6.2 Paint Results

In the "paint" scenario, 100 repetitions of the execution of the "paint" action for each of the algorithms were performed.

Table 4: "Paint" Evolutionary Approximation results: average results after 100 repetitions of the "paint" action for each of the proposed algorithms.

Algorithm	Evaluations	Discrepancy	Painted(%)
SST	539	7.25	94.4
PSO	583	12.06	91.44
AFFG-PSO	537	16.56	89.75
FI-PSO	441	20.13	87.88

Table 4 depicts the average results as a function of the used algorithm. Here, FI-PSO was again the most successful algorithm in terms of reducing the number of required evaluations. The number of required evaluations using this algorithm was reduced by 18% with a 6.52% less painted wall. The AFFG-PSO algorithm required the same number of evaluations than the SST algorithm with a 4.65% less painted wall. The discrepancy value was increased by more than the double for these two algorithms. Finally, the PSO algorithm required 8% more evaluations reducing the percentage of painted wall by 2.96% and increasing the discrepancy value a 66%.

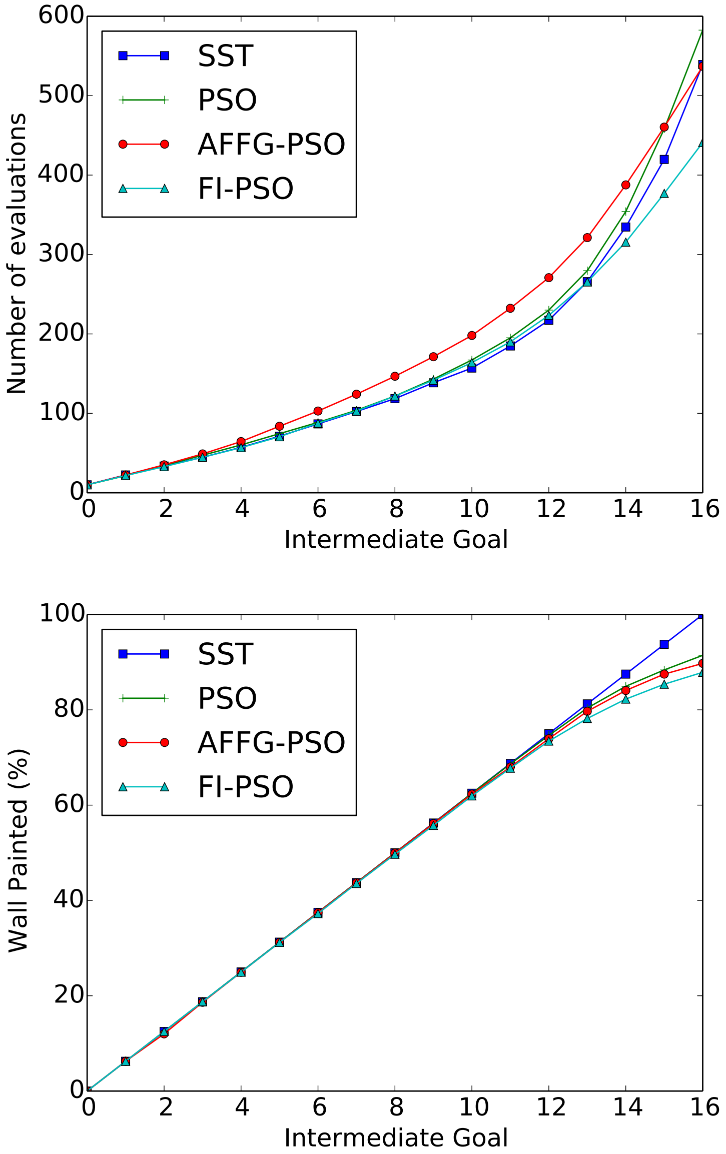

Figure 10: “Paint” Evolutionary Approximation results: results as a function of each intermediate goal defined by the generalized trajectory. At the top, the total number of required evaluations. At the bottom, total percentage of painted wall.

Fig. 10 depicts two different plots. The plot at the top is the total number of required evaluations. The plot at the bottom is the percentage of painted wall. Both plots are represented as a function of the intermediate goals presented in the generalized trajectory. In this action, the feature trajectory that encodes the action measures the percentage of painted wall.

The generalized trajectory is a straight line with a 0% percentage of painted wall at intermediate goal 0, and 100% at the last intermediate goal.

3.7 Constrained Genetic Algorithms Results

In these experiments, two different types of constraints were introduced: spatial and velocity constraints. A death penalty strategy was the strategy chosen to deal with constrained solutions. Solutions outside the non constrained space were assigned a Discrepancy value of ∞. These constraints were tested using the same actions as in the case of the Evolutionary Approximation experiments. These are the "paint" and the "wax" actions. The spatial constraint was introduced within the Cartesian search space of the action. This constraint is defined as the minimum bounding box around the solution space with a dilatation value. The results obtained are depicted as a function of this dilatation value. The velocity constraint was introduced in the joint space of the robot. This constraint is defined as the maximum joint velocity allowed for each of the joints. Different velocities thresholds were introduced. The results are depicted as a function of the value of this velocity threshold.

3.7.1 Wax Results

The "wax" action experiments consisted on 50 repetitions of the execution of the "wax" action using the CGDA framework. These number of repetitions were performed for both the spatial and velocity constraint.

Table 5: "Wax" spatial constraint experiment results: average results after 50 repetitions of the "wax" action for each of the dilatation values.

Dilatation [m]	0.01	0.05	0.1	0.2	0.3	∞
Evaluations	3212	3163	2993	4960	5722	9679
Discrepancy	465 (2)	503 (1)	471 (3)	312	331	274

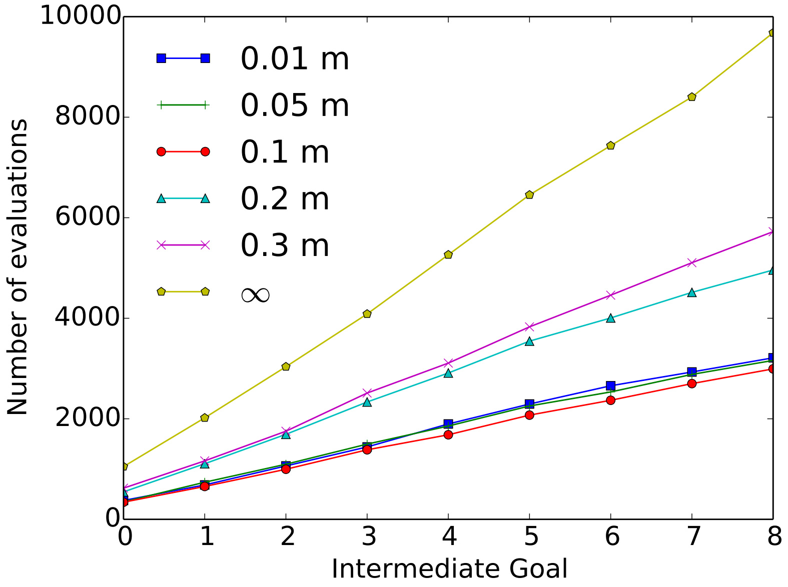

Figure 11: "Wax" spatial constraint experiment results: total number of required evaluations for the "wax" action execution as a function of the intermediate goals defined in the feature trajectory.

In the case of the spatial constraints, six different dilatation values were tested in the experiments: 0.01 m, 0.05 m, 0.1 m, 0.2 m, 0.3 m and ∞. Using a ∞ dilatation value is equivalent to the scenario of using the vanilla SST algorithm. The average results after 50 repetitions of the "wax" action, for each of the dilatation values, is depicted in Table 5. Here, the value in parenthesis (), in the Discrepancy row, depicts the number of obtained solutions that ended outside the constrained space. These solutions were assigned an ∞ discrepancy value and therefore not considered to be computed within the average of the discrepancy. In this scenario, the higher reduction in the number of required evaluations was obtained using a dilatation value of 0.1. The number of required evaluations was reduced by 69% increasing the discrepancy value by 71%. Lower dilatation values obtained worse results both in terms of number of required evaluations and discrepancy value. Higher dilatation values came with a reduction of the discrepancy value but also an increase in the number of required evaluations. In the case of a dilatation value of 0.2, the number of required evaluations was reduced by 49% with an increase of the discrepancy value of

14%. Fig. 11 depicts the total number of required evaluations as a function of the intermediate goals defined in the feature trajectory.

In the velocity constraints experiments, using the “wax” action, six different velocity thresholds were introduced: 5, 10, 20, 60, 70, 80 and ∞ [degrees/iteration].

Table 6: “Wax” velocity constraint experiment results: average results after 50 repetitions of the “wax” action for each of the velocity constraint values.

Max. Velocity [degrees/iteration]	5	10	20	60	80	∞
Evaluations	3591	4058	5723	6876	7349	9679
Discrepancy	540	483	331	346	330	274

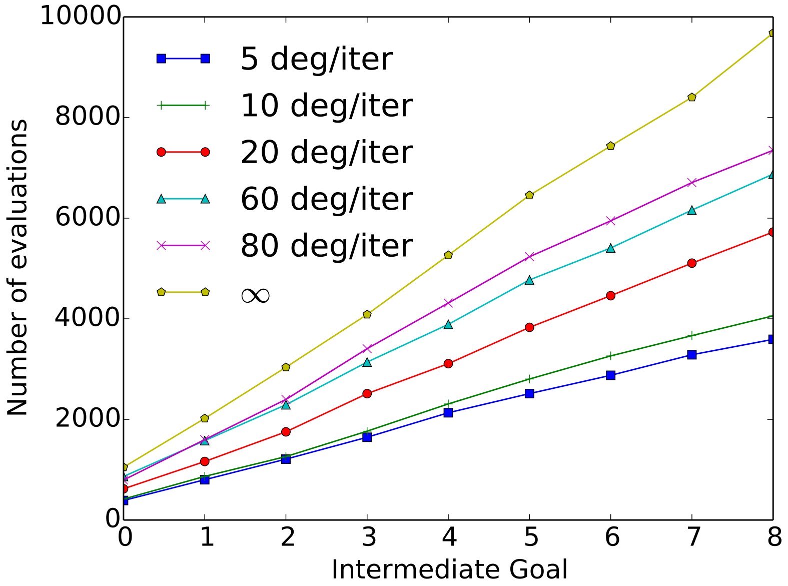

Figure 12: “Wax” velocity constraint experiment results: total number of required evaluations for the “wax” action execution as a function of the intermediate goals defined in the feature trajectory.

The average results for the velocity constraints experiments are depicted in Table 6. In these experiments, the higher reduction in the number of required evaluations was achieved with the lower velocity threshold (5). The number of required evaluations was reduced by 63% with an increase in the discrepancy value of 97%. Higher velocity thresholds required a higher number of evaluations but also obtained lower discrepancy values. For a threshold of 20, the number of required evaluations was reduced by 41% with an increase in the discrepancy of 21%. Fig. 12 depicts the total number of required evaluations as a function of the intermediate goals defined in the feature trajectory.

3.7.2 Paint Results

As in the Evolutionary Approximation experiments, 100 repetitions of the "paint" action were performed for each dilatation value. The same dilatation values as for the "wax" action were introduced.

Table 7: "Paint" spatial constraint experiment results: average results after 100 repetitions of the "paint" action for each of the spatial constraint values.

Dilatation [m]	0.01	0.05	0.1	0.2	0.3	∞
Evaluations	319	307	220	334	360	539
Discrepancy	193	17	5.7	7.3	6.3	7.3
Painted Wall (%)	64.06	89.58	95.13	94.38	94.56	94.44

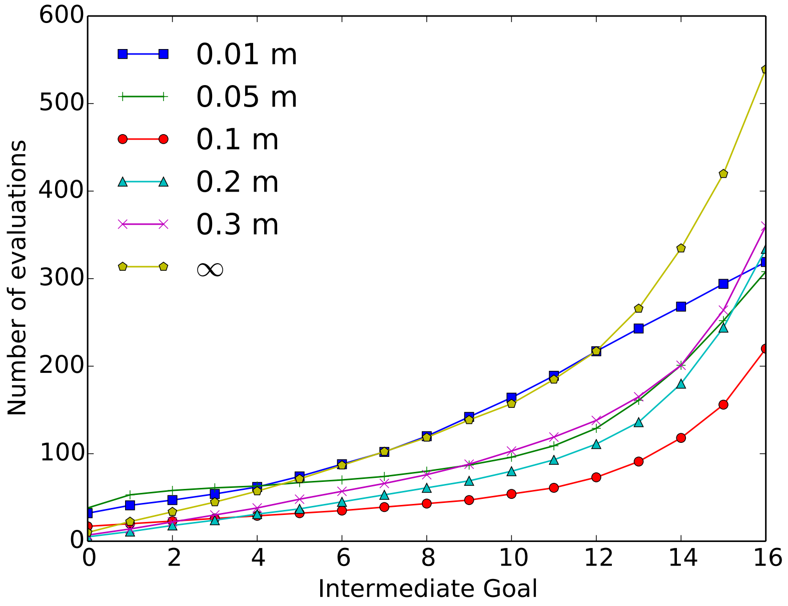

Figure 13: "Paint" spatial constraint experiment results: total number of required evaluations for the "paint" action execution as a function of the intermediate goals defined in the feature trajectory.

The results in Table 7 were the average obtained for these experiments. Here, the minimum number of required evaluations was obtained with a dilatation value of 0.1. The number of required evaluations was reduced by 60% with also a decrease in the discrepancy value of 22% with respect the SST algorithm and a 0.7% more painted wall. Fig. 13 depicts the total number of required evaluations as a function of the intermediate goals defined in the feature trajectory.

Table 8: "Paint" velocity constraint experiment results: average results after 100 repetitions of the "paint" action for each of the velocity constraint values.

Max. Velocity [degrees/iteration]	20	60	80	100	120	∞
Evaluations	543	557	572	527	529	539
Discrepancy	24.8	12.8	10.4	8.1	8	7.3
Painted Wall (%)	87.06	90.21	92.68	93.75	93.88	94.44

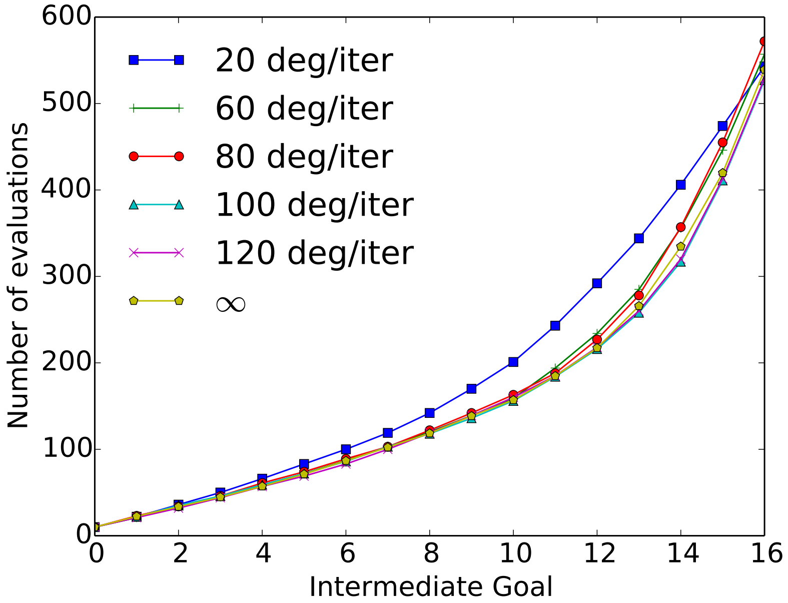

Figure 14: "Paint" velocity constraint experiment results: total number of required evaluations for the "paint" action execution as a function of the intermediate goals defined in the feature trajectory.

In the case of the introduction of velocity constraints within the "paint" action, six velocity constraint values were chosen: 20, 60, 80, 100, 120 and ∞. Again, an ∞ constraint value was equivalent to the original non constrained SST algorithm. The average results obtained after the 100 repetitions for each of the constraint values are depicted in Table 8. In these experiments, a relevant reduction in the number of required evaluations was not obtained. The higher reduction was of a 2% obtained for the 120 velocity constraint. The percentage of painted wall, however, decreased to a minimum of 87.06% (7.7% lower with respect SST) with the introduction of more restrictive constraints. Fig. 14 depicts the total number of required evaluations as a function of the intermediate goals defined in the feature trajectory.

3.8 Conclusions

In this chapter, a study of different methods to reduce the number of required evaluations in evolutionary algorithms is proposed. Real world scenarios are highly dynamic environments that are constantly changing. Reducing the number of required evaluations allows the CGDA framework to reduce computational times and in some situations to perform real world evaluations. The goal is to give the framework the ability to achieve an online adaptation to environment changes. Two different approaches are presented: Evolutionary Approximations and constrained genetic algorithms. Evolutionary Approximations can be introduced as a way to reduce evolutionary algorithms complexity. These techniques can be classified as a function of where these approximations are introduced. Two Evolutionary Approximation algorithms have been tested in the experiments: AFFG-PSO and FI-PSO. The first one, AFFG-PSO, being an original contribution. In the second approach, referred as constrained genetic algorithms, two different types of constraints were introduced and studied within the CGDA framework: these are the spatial and velocity constraint. The spatial constraint was introduced in the Cartesian search space of the robot. The velocity constraint was introduced in the joint space of the robot. A death strategy was implemented to deal with solutions outside the non constrained space. Both of these approaches, Evolutionary Approximation and constrained genetic algorithms, were tested in experiments with the CGDA framework and the humanoid robot TEO. Two different actions, "paint" and "wax", were introduced to test the performance of these approaches. The experiments measured the number of required evaluations and the action performance for each of the proposed methods.

Introducing these methods successfully reduced the number of required evaluations for both of the actions. In some configurations, this reduction came with a decrease in the performance of the action execution. In others, the performance was kept the same or even increased. Reductions in the number of required evaluations of 60% were achieved with certain configurations. In the case of the Evolutionary Approximation algorithms, FI-PSO obtained the best results for the "wax" action. For the "paint" action, FI-PSO obtained a higher reduction in the number of required evaluations than AFFG-PSO, but with a worse performance. In the case of constrained genetic algorithms, the velocity constraint obtained better results than the spatial constraint for the "wax" action. In the case of the "paint" action, the spatial constraint achieved a 60% reduction in the number of required evaluations while increasing the performance.

Two main conclusions can be extracted from these results. The first one is that although the number of required evaluations is highly reduced for some configurations, most of the times, this number is still high enough to be too costly to directly perform these evaluations in real world environments. The second conclusion is that the performance of these methods varies as a function of the target executed action. Some tuning has to be performed for each action. This limits the generalization capabilities of the framework. In the following chapter, a new evolutionary strategy to allow CGDA to work in real world scenarios without these limitations is proposed. This new strategy does not aim at reducing the number of evaluations, but rather to introduce an online generated model of the environment.

4. ONLINE EVOLVED TRAJECTORIES

In Continuous Goal-Directed Actions (CGDA), robot joint trajectories for the execution of actions are generated using evolutionary algorithms. These algorithms require high numbers of evaluations to converge. These evaluations are performed in simulations using models of the environment. This is a computational expensive process that usually takes a relevant amount of time to converge. In these scenarios, robot joint trajectories obtained with evolutionary algorithms may be no longer valid after being generated. In the previous chapter, the introduction of different methods to reduce the number of required evaluations has been studied. Although achieving relevant reductions, this number was still too large to directly perform the evaluations in real world scenarios.

In this chapter, a different approach is introduced. Here, the introduction of online evolutionary strategies is proposed. In online evolutionary strategies, real world executions are introduced within the planning loop. This planning loop iterates between the goals defined in the feature trajectory to generate the corresponding robot joint trajectories. In classical CGDA evolutionary strategies, generated robot joint trajectories are executed after this planning loop. In online evolutionary strategies, for each iteration of the planning loop, a real world execution of the generated robot joint trajectory is performed. An observation step is then executed and a new model of the environment is generated. The model is updated to contain the last changes of the environment in order to produce valid robot joint trajectories. In this chapter, Online Evolved Trajectories (OET)

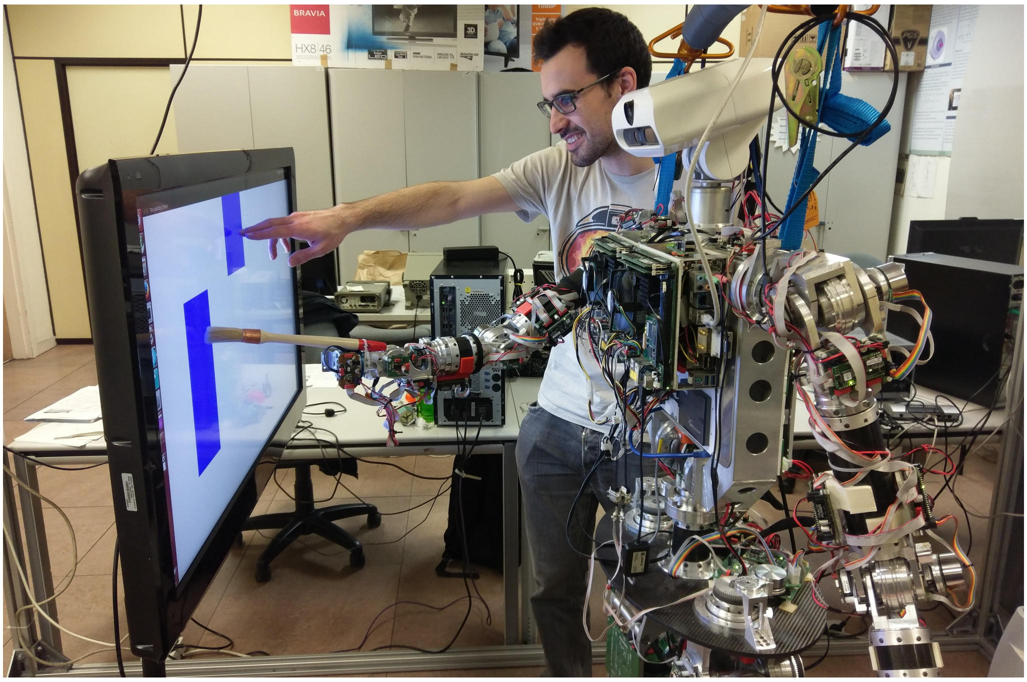

Figure 15: The OET algorithm allows introducing changes in the environment during execution. In the image, a collaborative execution of the“paint” action is depicted.

is proposed as an online evolutionary strategy for the CGDA framework. OET drastically reduce computational times between real world executions. This allows CGDA to be executed in highly dynamic real world scenarios. This is the case of human robot collaborative actions such as the one in Fig. 15. The performance of the OET strategy is compared with the FTE and IET strategies presented in previous chapters. The experiments are performed using two actions: “paint” and “iron”. The “iron” action is presented as a new action defined in the scope of this work. Both of these actions were chosen to include vision, kinesthetic and force features.

4.1 Online Evolutionary Strategies

In classic evolutionary CGDA strategies, actions are only executed in the real world after the generation of the full robot joint trajectory. These strategies are considered offline, due to having the planning stage detached from the real world executions. This planning stage is in charge of the generation of the robot joint trajectories that achieve all the intermediate goals defined in the feature trajectory. A general layout of an offline evolutionary strategy is presented in the pseudo code in Algorithm 5.

Algorithm 5 Offline Evolutionary Strategy

```
procedure OFFLINE
    while not planning_termination_conditions do
        mental_process_loop
    end while
    motor_execution($U$)
end procedure
```

In this thesis, online evolutionary strategies is proposed as an alternative approach to deal with highly dynamic environments. In online evolutionary strategies, the real world execution step is introduced within the planning stage. Every time the robot joint trajectory required to reach any intermediate goal is generated, a real world execution of this part of the trajectory is performed. This increases the frequency between real world executions, allowing the framework to adapt to changes produced in the environment. A general layout of an online evolutionary strategy is presented in Algorithm 6.

Algorithm 6 Online Evolutionary Strategy

```
procedure ONLINE
    while not planning_termination_conditions do
        mental_process_loop
        motor_execution($U_j$)
    end while
end procedure
```

Here, a motor execution step is executed for each intermediate robot joint trajectory U_j generated. In classical offline evolutionary strategies, this execution is only performed after the generation of the full robot joint trajectory U.

4.2 The Online Evolved Trajectories algorithm

Online Evolved Trajectories (OET) is proposed in this thesis as a concrete application of the online evolutionary strategies approach. This algorithm introduces the idea of online evolutionary strategies with the introduction of a perception and localization step. Algorithm 7 depicts the pseudocode of the OET algorithm.

Algorithm 7 Online Evolved Trajectories (OET)

```
procedure OET($X$)
    $individuals$ ←initialize
    while not $oet\_termination\_conditions$ do
        $P_t$ ←sensor_perception
        $j$ ←localization($P_t$)
        while not $termination\_conditions$ do
            for each $individual$ do
                $U_{j+1}$ ←evolve($DoF$)
                $O_{j+1}$ ←mental_execution($U_{j+1}$)
                $f$ ←mental_recognition($O_{j+1}$,$X_{j+1}$)
            end for
        end while
        motor_execution($U_{j+1}$)
    end while
end procedure
```

The *sensor_perception* and *localization* steps are two new steps introduced for the OET algorithm. These steps are in charge of updating the state of the environment model and the current feature trajectory.

4.2.1 Perception Step

In the sensor perception step, the robot sensors are used to extract the current state of the environment encoded as the state of the features defined in the generalized feature trajectory. An m-dimensional feature vector P_t, defined as in Eq. 4.1, is generated each time step t for the current environment state.

$$P_t = [p_{0t}, p_{1t}, p_{2t}, p_{3t}, p_{4t} \ldots, p_{mt}]^T \tag{4.1}$$

4.2.2 Localization Step

In the Localization step, the vector P_t is used to find the current state of the executed trajectory in the generalized trajectory. This current state is defined as the intermediate goal X_j that is more similar to the current environment state P_t as defined in Eq. 4.2.

$$j = \underset{j \in [j_{prev}, n]}{\arg\min} \ (\|P_t - X_j\|_p) \tag{4.2}$$

Where j_{prev} corresponds to the last achieved intermediate goal and p is the order of the norm introduced, preferably fixed as 2 as the Euclidean L2 norm.

4.3 Experiments

Experiments over three different evolutionary strategies were performed: Full Trajectory Evolution (FTE), Incrementally Evolved Trajectories (IET), and the Online Evolved Trajectories (OET). The goal was to compare the performance of OET with respect the base strategies proposed with CGDA. These three strategies were tested with two different actions: the already defined “paint” action introduced in the experiments in section 3.5; and a new “iron” action defined for these experiments. These results were also compared with the ones obtained via Gaussian Mixture Regression as proposed by Calinon et al. (9). The robot platform selected was the humanoid robot TEO from Universidad Carlos III de Madrid. The Individual Evolution (IE) strategy was not introduced due to the inherent issue explained in section 2.4.2. This issue affects time dependent actions such as the “paint” action proposed in the experiments.

For the experimental setup, in order to execute the “paint” action, a paintbrush was attached to the left arm end-effector of the robot. In the case of the “iron” action, a real iron was attached to the right arm end-effector. For the demonstrations, the 6 DoF of the corresponding arm (left for “paint”, right for “iron”) were used. The other robot joints (torso, legs, head...) were kept static. The demonstrations were performed via kinesthetic learning setting all the joints of both of the robot arms into gravity compensation mode. The demonstrator was asked to execute each of the actions separately using only the robot arms. In the case of the “paint” action, the encoded feature was the percentage of painted wall. The goal of the “iron” action was to descend the iron to the ironing board, apply a force of approximately 30 N, and then ascend again. Two sets of features were measured for this action. First, the Cartesian position of the robot end-effector was computed using the CUI absolute encoders in the robot joints and forward kinematics. Second, the force exerted in the robot end-effector was measured using the JR3 force/torque sensor equipped in the right wrist of the robot. Four demonstrations were performed for each of the actions.

Mental executions were performed inside a simulated environment using OpenRAVE. YARP was introduced for the internal communications. The strategies were implemented using the ECF C++ framework (54). In order to have a common baseline with previous experiments, SST was chosen as the evolutionary algorithm for the evolutionary algorithm step. The code to run the experiments was open-sourced and made available online[1].

---

[1] `https://github.com/roboticslab-uc3m/xgnitive`

The SST number of individuals was set to a population of 10. The tournament size was set to a value of 3. A 60% mutation probability was introduced. Each individual joint value was limited to the range [-15, 100]. This range corresponds to the arm joint limits expressed in degrees. The termination conditions for FTE were set to reach a maximum number of 300 iterations or reaching 75 iterations without improving the fitness value. In the case of the IET strategy, these termination conditions were set to $300/n$ total number of iterations or $75/n$ iterations without improvement, where n is the total number of intermediate goals. For the OET algorithm, these numbers were set to $300/otc$ total iterations or $75/otc$ iterations without improvement, where otc is the maximum number of allowed evolutionary algorithm executions in the OET algorithm. The otc parameter was set to be equal to n. With these parameters, the total number of possible evaluations was the same for the three evolutionary strategies.

4.4 Results

In addition to the Discrepancy value and the number of Evaluations, a new parameter was introduced to measure the results of this chapter. This new parameter is the Real Iteration Time (RIT). The RIT measures the time between two real world executions as defined in Eq. 4.3. This parameter measures the update frequency of the environment model. The results obtained in the following sections are the average of three repetitions of the action for each of the strategies proposed.

$$RIT = t_j - t_{j_{prev}} \tag{4.3}$$

4.4.1 Paint Results

The average duration defined by the "paint" demonstrations was $D_{time} = 130.2\ s$. The selected time interval for the action was $T_{min} = 10\ s$ with $n = 13$ intermediate goals. Each of these demonstrations were performed following a different geometrical path. Fig. 16 depicts the "paint" demonstrations as well as the obtained trajectory using Gaussian Mixture Regression. The trajectory obtained with the Gaussian Mixture Regression model achieves a 43.75% of painted wall.

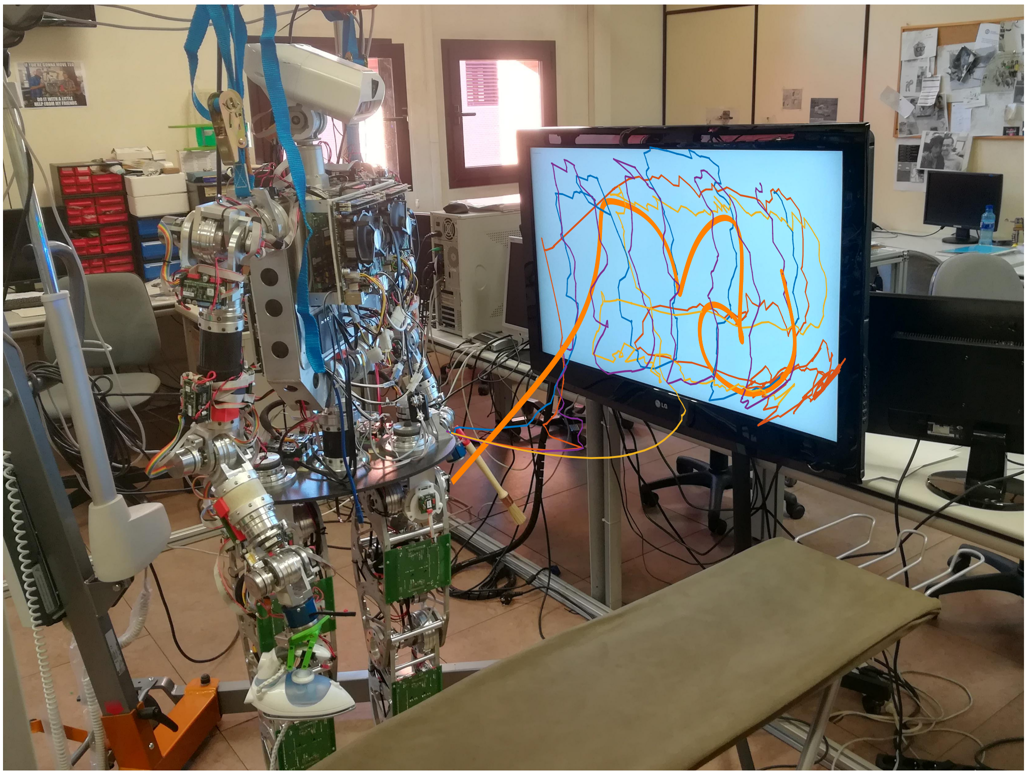

Figure 16: Resulting trajectories for the demonstrations of the "paint" action. The orange thick line depicts the result of executing the Gaussian Mixture Regression method ($K = 7$, $T = 600$) using these demonstrations.

Table 9 depicts the results obtained using CGDA with each of the strategies proposed. The results depict critical reductions in terms of RIT between strategies. The RIT value obtained in the experiments goes from 272.3 seconds with FTE and 143 seconds with IET to 4 seconds for the OET strategy. This reduction in RIT did not come with a reduction in the

Table 9: "Paint" OET experiment results: average and standard deviation results after 3 repetitions of the "paint" action for each of the proposed strategies.

	Evaluations		Discrepancy		RIT [s]		Painted Wall [%]	
Strategy	μ	σ	μ	σ	μ	σ	μ	σ
FTE	1716	231.80	49.48	7.40	272.3	68.48	85.4	3.6
IET	1153	161.65	54	25.36	143	25.87	72.9	15.72
OET	1603	20.82	40.19	3	4	0.6	89.58	3.6

performance. The performance in terms of Discrepancy and percentage of painted wall was improved in the case of the OET algorithm. The generalized feature trajectory introduced for the action as well as the features trajectories obtained for each of the strategies are depicted in Fig. 17.

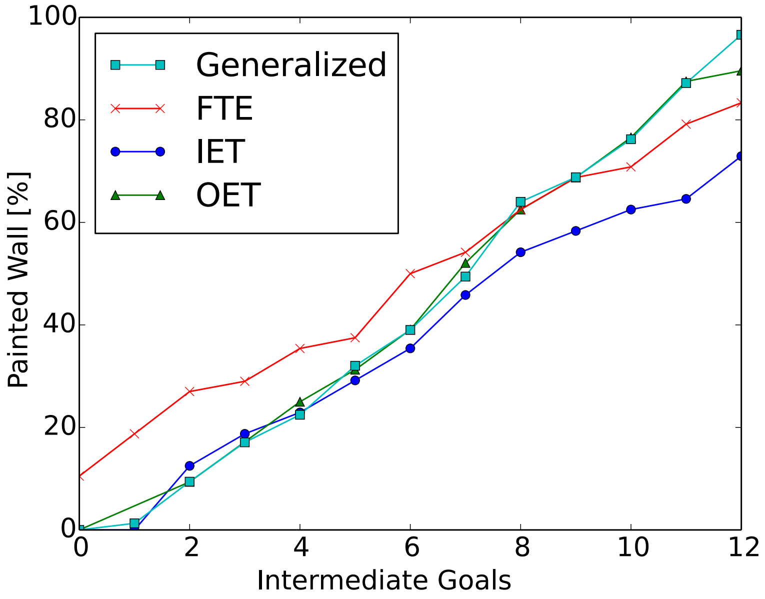

Figure 17: "Paint" OET experiments results: generalized feature trajectory for the "paint" action compared to the features trajectories obtained for each of the algorithms.

4.4.2 Iron Results

The average demonstration time obtained for the "iron" action was $D_{time} = 28.1\ s$. The selected time interval was $T_{min} = 3\ s$ with $n = 9$ intermediate goals. Fig. 10 depicts the trajectories followed by the demonstrations as well as the resulting trajectory using the Gaussian Mixture Regression method. Although geometrically accurate, the force exerted by the iron using the trajectory computed with Gaussian Mixture Regression was close to zero.

The results obtained in the "iron" experiments are depicted in Table 10 and Fig. 18. These results are in line with the results obtained for the "paint" action. The RIT was reduced to 1.4 seconds for the OET strategy compared to the 30.3 seconds required for IET and the 2481 seconds required for the FTE strategy. The performance in terms of Discrepancy value was also improved with the introduction of the OET algorithm.

Table 10: "Iron" OET experiments results: average and standard deviation results after 3 repetitions of the "iron" action for each of the proposed strategies.

	Evaluations		Discrepancy		RIT [s]	
Strategy	μ	σ	μ	σ	μ	σ
FTE	3010	0	0.7	0.09	2481	1.73
IET	1588	113.74	0.59	0.05	30.30	2.69
OET	1010	400.37	0.30	0.07	1.44	0.16

4.5 Conclusions

Online evolutionary strategies has been proposed as a new evolutionary approach for the execution of actions in real world environments within the CGDA framework. This new strategy does not aim at reducing the number of evaluations, but rather to introduce an online generated model of the environment. This approach proposes the idea of introducing real motor executions within the planning loop. These real motor executions

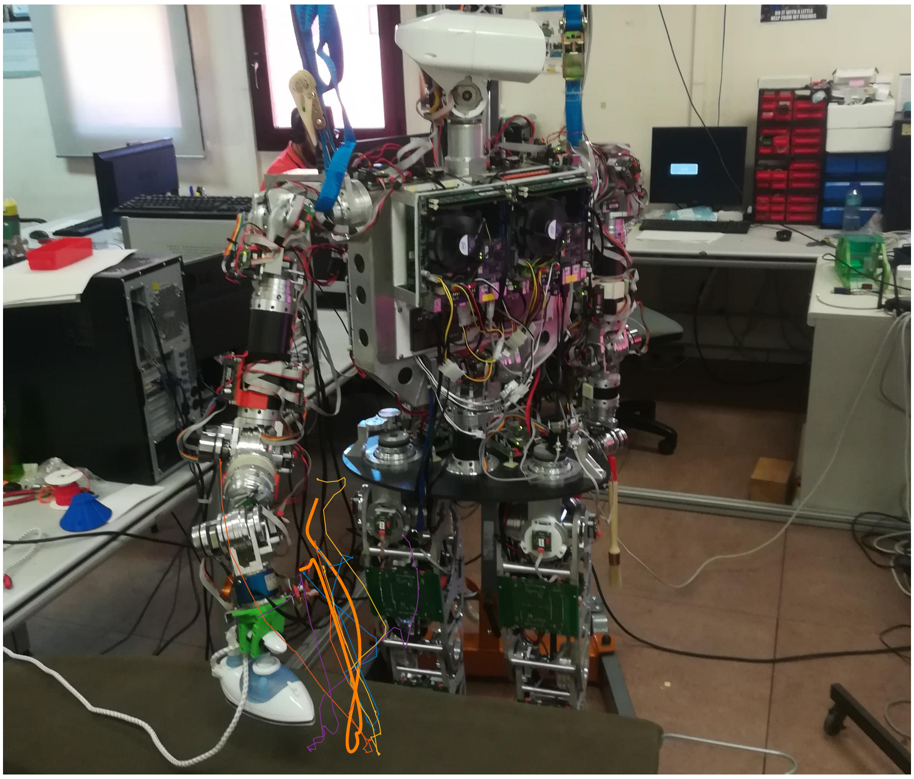

Figure 18: Resulting trajectories for the demonstrations of the “iron” action. The orange thick line depicts the result of executing the Gaussian Mixture Regression method ($K = 5$, $T = 150$) using these demonstrations.

are performed while the robot joint trajectory is being generated. After these real world executions, the new state of the environment is transferred to the simulation model. The algorithm OET has been proposed as a concrete application of online evolutionary strategies. This algorithm introduces a localization and an observation step to update the environment and the executed feature trajectory. To test the performance of OET, experiments using two different actions were performed. These two actions are the “paint” and “iron” actions. The “paint” action is the same action as the one presented in previous chapters. The “iron” action is a new action introduced for the experiments of this chapter.

The results of the experiments were compared with the ones obtained using IET and FTE. The resulting trajectory was compared with the one obtained using Gaussian Mixture Regression. A new parameter called RIT was introduced to measure the update frequency of the environment model.

Lower RIT implies a higher model frequency update allowing the robot to work with higher dynamic environments. In the results of the experiments, OET critically reduced the RIT required with respect previous strategies. This value was reduced from minutes to a few seconds. The more drastic reduction was obtained for the "iron" action. Here, the FTE strategy obtained an RIT of 2481 seconds while OET obtained an RIT value of 1.4 seconds. This RIT value is at the level of the ones observed in humans (55). This reduction was obtained without compromising the performance. The introduction of OET improved the overall performance for both of the actions.

Future works introducing OET are related to CGDA applications with highly dynamic environments. One of this applications is the case of human-robot collaborative actions. Here, the robot has to adapt to the changes produced by the human while executing the action. In the case of the "paint" action, for example, it is important that the robot does not paint the same areas already painted by the collaborator as in Fig. 15.

CONCLUSIONS OF PART II: CONTINUOUS GOAL-DIRECTED ACTIONS

In this part of the thesis, CGDA, a learning framework for action generalization (3), has been introduced. As an original contribution, two different approaches to improve the CGDA execution within real environments have been proposed.

The first approach was the introduction of Evolutionary Approximations and constrained genetic algorithms. These two methods are introduced to reduce the number of required evaluations in evolutionary algorithms. The goal is to reduce computational times to allow faster adaptations or, in some contexts, to directly perform real world evaluations in the environment. The results show reductions in the number of evaluations higher than 60% for some configurations. For most scenarios, these are still too many evaluations to be directly performed in the real world. Additionally, these methods require some additional action specific tuning, which contrasts with the initial goal of achieving action generalization.

The second approach took a different perspective. Online evolutionary strategies was proposed as a new paradigm for the implementation of evolutionary algorithms within the CGDA framework. The idea of this approach was to introduce real motor executions within the planning loop. Every time a robot joint trajectory for an intermediate goal is generated, it is executed and the environment is updated. OET was proposed as a concrete application of this paradigm. The results show a critical reduction in the RIT with respect previous strategies. For one of the actions, this reduction allowed CGDA to reach human-level reaction times. This is a critical improvement with respect previous strategies where RIT values were over minutes.

These results come at the same time of the emerging of Deep Learning as an important force within the Machine Learning community. Deep Learning frameworks are being proposed achieving human-level performance in many different areas. In the next part of this thesis, the introduction of Deep Learning is proposed as a way to achieve action generalization in robotics. Two different methods will be studied: Style Transfer and Deep Reinforcement Learning (DRL). A new framework introducing these two methods will be proposed. The goal is to give the robot the ability to introduce different styles in the same base action. The same action can be adapted to work in different contexts and environments with the introduction of different Styles. At the same time, DRL is introduced as a way to directly implement robot controllers. The introduction of DRL allows robot actions to be applied in complex dynamic environments.

Part III

Neural Policy Style Transfer (NPST)

5. REINFORCEMENT LEARNING AND STYLE TRANSFER

The melding of Reinforcement Learning and Style Transfer techniques is proposed in this thesis as a way to introduce Style Transfer algorithms within robotic actions. The introduction of Reinforcement Learning architectures, specifically Deep Reinforcement Learning (DRL) architectures, has been proposed in the robotic scientific community as a way to directly generate robot controllers. Style Transfer is introduced as a way to provide robots the ability to transfer different Styles to base actions. The goal is to allow the adaptation of actions to different contexts and environments by transferring different Styles. In this chapter, an introduction to the Reinforcement Learning and Style Transfer background is proposed.

5.1 Reinforcement Learning

Reinforcement learning is a learning framework that is at the same time different from supervised learning and unsupervised learning (18). In reinforcement learning, the user introduces a reward function to implicitly define the goal of the problem. This reward function assigns a reward to each state. For finite tasks, the goal of the agent is to maximize the total obtained reward over a full episode. The agent behavior is defined using a policy function. The policy function defines the robot action as a function of the current state.

A problem in reinforcement learning can be defined using the tuple $T = \{S, A, \gamma, R\}$: where S is the state space defined for the problem; A is the action space available to the agent within this state space; γ is a discount factor introduced to deal with future rewards; and R is the reward function of the problem. This tuple is usually defined by the user. The reinforcement learning algorithm is then in charge of generating the optimal policy $\pi^*(s)$ using this tuple. This policy defines the behavior of the agent. A value function $V_\pi(s)$ –also known as state-value function– can be defined as the expected total reward obtained following π as a function of the current state s. Similarly, a value function $Q_\pi(s,a)$ –also known as action-value function, Q-value function, or simply Q-function– can be defined as the expected total reward obtained following π as a function of the current state s and the current action a. $Q_\pi(s,a)$ can be defined as in Eq. 5.1.

$$Q_\pi(s,a) = E_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle| S_t = s, A_t = a\right] \tag{5.1}$$

where t is any time step and $E_\pi[]$ is the expected reward following π.

5.1.1 Q-Learning

One of the most successful approaches in reinforcement learning are Temporal Difference methods. Temporal Difference methods are model-free methods that merge the ideas of Monte Carlo and Dynamic Programming methods. In Monte Carlo methods, the update of the value function is only produced at the end of the episode. In Temporal Difference methods, each time step, an estimation of the error, using the obtained reward R, is computed and the value function is updated.

One of the most popular methods within the Temporal Difference approach is Q-learning. Q-learning is an off-policy method where the obtained value function $Q(s,a)$ is independent of the policy used during training. The only requirement for the training policy is that all action-state pairs are visited. Upon convergence, the optimal policy π^* can be defined as the greedy policy that maximizes the Bellman equation, via the update rule defined in Eq. 5.2.

$$Q(S_t,A_t) = Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)] \tag{5.2}$$

where α is the learning rate.

5.1.2 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) frameworks have gained popularity by proposing the introduction of deep neural networks within a reinforcement learning architecture. DRL allows the generation of complex robot controllers in the form of a deep neural network. Different applications have been recently proposed using DRL in robotics (56, 57, 58).

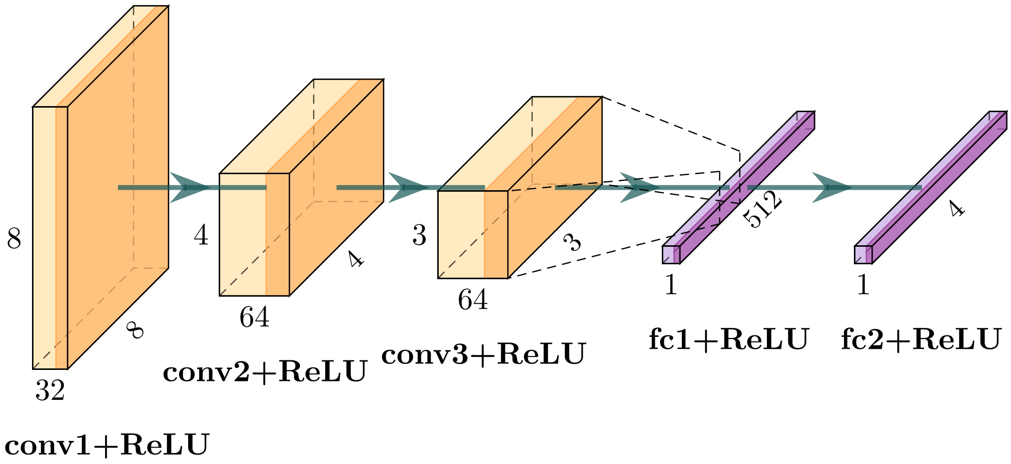

Figure 19: Deep neural network architecture similar to the one proposed by Mnih et al. (2) for the introduction of DRL in Atari games.

The idea of DRL is to use a deep neural network to define an element of a Reinforcement Learning problem. Mnih et al. (2) proposed the introduction of a deep neural network to define the value (specifically, the Q-value) function. This Deep Q-Network (DQN) architecture was trained for playing Atari video games at human-level. An architecture similar to the one proposed by Mnih et al. is depicted in Fig. 19. The output of the deep neural network is the Q-value function $Q(S_t, A_t)$ corresponding to the state S_t passed as input. The results obtained in this paper greatly attracted the attention of the scientific community towards DRL.

5.1.3 Continuous action spaces

Classical reinforcement learning methods are designed to work using Markov Decision Processes. In Markov Decision Processes, actions are encoded using a discrete action space. At the same time, robotics applications work with continuous and high dimensional action spaces. The introduction of a greedy policy in a continuous action space would require to perform an optimization problem every step. Deep Deterministic Policy Gradient (DDPG) was proposed by Lillicrap et al. (59) as a way to introduce DRL within continuous action spaces. In DDPG, the policy is defined as a parametric function $\mu(s, \theta)$ using an additional deep neural network. This policy is trained to maximize the output of the Q-value function. Two deep neural networks are then defined, one for the Q-function and one for the policy. An actor-critic architecture is introduced to implement these two networks. The actor corresponds to the policy, while the critic is the Q-function.

Twin Delayed Deep Deterministic Policy Gradient (TD3) (60) was later proposed as a more stable new version of DDPG. TD3 improves DDPG with the introduction of Double Q-learning (61), delayed policy updates, and target policy smoothing.

5.2 Style Transfer: Content and Style

The idea of Style Transfer is to transfer *how* a certain element is to a different element without changing *what* this different element is. In natural language processing, the *how* can be defined as the specific selection of words, and the *what* as the actual meaning. In computer vision with fine arts, the *how* would be the artistic technique of the painter, while the *what* would be the objects in the painting. In animation, the *how* could be defined as the emotion of the movements, while the *what* is the target defined for the movement. The definition of these two concepts is the base of any Style Transfer application. The *what* is typically referred as the Content, while the *how* is referred as the Style. The concrete definition of these two concepts depends on the authors and the application.

The terms Style and Content were first introduced in the area of computer vision, concretely in the context of optical character recognition (62). The first works introducing Style Transfer in motions were part of the computer animation scientific community (63). These works focus on the introduction of signal processing techniques to perform the Style transfer step (64, 65). Recent works introduced more advanced techniques, such as multilinear model design (66). In robotics, the introduction of Style has been tied to the introduction of emotions. In order to achieve this, some works have included methods such as the introduction of Laban movement systems (67) or cost functions (68). Other works have proposed the introduction of Generative Adversarial Networks (GAN) (69). Recently, Wang et al. (70) proposed the introduction of a GAN Style Transfer framework for Avatar drawing using a robot manipulator. In Liu et al. (71) a GAN based Style Transfer framework is introduced to transfer the Style of a reference image to a video.

5.2.1 Neural Style Transfer

Neural Style Transfer was initially proposed by Gatys et al. (1) as an application of Deep Learning within a Style Transfer architecture. In Neural Style Transfer, a deep neural network is introduced to define the Content and Style. The Content is encoded as the high level outputs extracted from the last layers of the network. The Style is encoded as the Gram matrix of the low level outputs corresponding to the first layers of the network. The Style Transfer step is designed as an optimization process. The loss function of this optimization process is defined as the sum of the Content and Style losses. Gatys et al. introduced the VGG-19 network (14) as the deep neural network to extract the Content and Style. This VGG-19 network is a popular network designed for image classification, and was pre-trained with the ImageNet (72) dataset for the application. The weights of the network remain constant through all the iterations of the optimization process.

The introduction of deep neural networks adds a new layer of abstraction to Style Transfer frameworks. This eased the integration of Style Transfer within a wider range of different areas. Applications introducing Neural Style Transfer have been proposed in areas such as: computer vision (73, 74), natural language processing (75, 76), and fixed trajectories for animated figures as proposed by Holden et al. (12). In this work, Holden et al. proposed the introduction of Autoencoders as an alternative to the VGG network. This was proposed as a solution to the difficulty of having a deep neural network for motion classification. An Autoencoder is a neural network architecture integrated by two stages: the encoder stage and the decoder stage. The encoder stage generates a low dimensional representation of the input: the latent space. For a simple network, this follows Eq. 5.3. The decoder stage takes the output of the encoder and regenerates the input, as in Eq. 5.4. This regeneration is achieved via self-supervised training, where the output is forced to match the input.

$$A(X) = ReLU(\Psi(X * W_0 + b_0)) \tag{5.3}$$

$$\tilde{A}(H) = \tilde{\Psi}(H) * \tilde{W_0} - b_0 \tag{5.4}$$

where X is the input network; W_0 is the weight vector of the encoder; b_0 is the layer bias; and Ψ is a pooling operation. For the decoder: H is the encoder output; $\tilde{\Psi}$ is an inverse pooling operation; and $\tilde{W_0}$ is the weight matrix of the decoder.

The Content loss can be defined as the difference between the outputs of the encoder when passing the Content and when passing the generated motion following $L_{content} = \|A(C) - A(G)\|$. The Style loss is defined as the difference between the Gram matrix of the decoder outputs when passing the Style and when passing the generated motion following $L_{style} = \|Gm(A(S)) - Gm(A(G))\|$. The total Style Transfer loss is the sum of these two individual losses as in Eq. 5.5.

$$L_{st} = w_c L_{content} + w_s L_{style} \tag{5.5}$$

where w_c and w_s are constants defining the weight of these losses; C is the Content motion; S is the Style motion; G is the generated motion; and $Gm()$ corresponds to the Gram matrix operator.

5.2.2 Conclusions

Reinforcement learning is a learning framework where the goal of the problem is encoded using a reward function defined by the user. The output of reinforcement learning is a policy that maximizes the reward obtained by the agent over a full execution of the problem. The policy is a mathematical function in charge of mapping states with actions. A value function

$V_\pi(s)$ can be defined as the expected total reward obtained following policy π from any state s. Q-learning is an off-policy reinforcement learning algorithm where the goal is to find the Q-function $Q(S_t, A_t)$. DQN introduces Deep Learning within a reinforcement learning architecture. TD3 is proposed as a state of the art method to apply DQN within continuous action spaces.

Style Transfer has gained popularity after the introduction of Neural Style Transfer by Gatys et al. (1). The idea of Style Transfer is to transfer the Style of one input to a different input without changing the Content of this one. Neural Style Transfer introduces the idea of Deep Learning within an Style Transfer framework. Here, a pre-trained deep neural network is used to extract the Content and the Style. The introduction of deep neural networks introduces an additional layer of abstraction allowing Neural Style Transfer to be applied to a wider range of applications. The introduction of Autoencoders was proposed by Holden et al. (12) as an alternative to the VGG network proposed by Gatys et al. for the extraction of Content and Style.

The goal of this chapter was to provide to the reader with the required background in reinforcement learning and Style Transfer to understand the algorithms proposed in the following chapters.

6. NPST IN DISCRETE ACTION SPACES

In this chapter, we scale the concept of Style Transfer up to be used with control policies using a Deep Reinforcement Learning architecture. Deep neural networks architectures are introduced to define the Content and Style using Q-value functions. Two different sets of demonstrations are introduced: one for the Content and another for the Style. Inverse Reinforcement Learning, which will be described in section 6.1, is used to define the reward function corresponding to these networks. The Content action is defined using user demonstrations of the target action. The Style is extracted from a different set of demonstrations and defined as a secondary task regarding how an action can be performed. Different Styles can be encoded using different sets of demonstrations. The Neural Policy Style Transfer (NPST) algorithm is proposed to transfer the Style of one policy to another, without changing the Content of the latter. The NPST algorithm generates a new policy using the Content policy and the Style policy as input. Two sets of experiments are introduced in this chapter: Catch-ball game experiments, inspired by classical Deep Reinforcement Learning problems using Atari games; and Grid-world paint experiments, which are real world experiments using the full-sized humanoid robot TEO and introducing an action similar to the “paint” action presented in previous chapters. Fig. 20 depicts an action demonstration introduced for the Grid-world paint experiments of this chapter.

NPST is proposed as a way to improve the generalization capabilities of robot actions. Different Styles can be introduced in a base Content

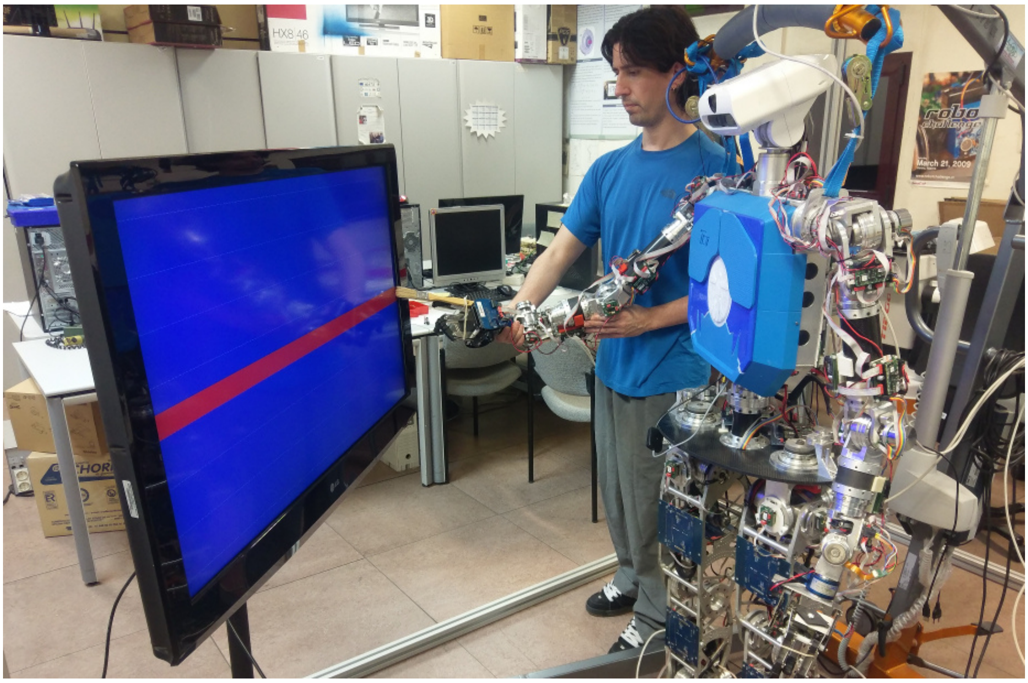

Figure 20: Grid-world paint scenario demonstration setup.

action as a way to achieve action adaptation. The same Content action can be used in different contexts and situations using different recorded Styles. This chapter proposes the following contributions that can derive in a wider range of future applications:

- A framework that allows the introduction of Inverse Reinforcement Learning (IRL) algorithms to extract Content and Style from user demonstrations.
- The NPST algorithm as a way to perform Style Transfer between policies defined using DQN.
- Results within two different experimental scenarios: the Catch-ball game scenario based on classical DRL problems using Atari games; and the Grid-world paint scenario using the real humanoid robot TEO and based on the "paint" action proposed on previous works of the authors.

6.1 Inverse Reinforcement Learning

The implementation of a proper reward function is a critical step that defines the agent behavior in reinforcement learning. Inverse Reinforcement Learning (IRL) is proposed as a way to extract this reward function R using a set of m user demonstrations defined as in Eq. 6.1.

$$E = \{s_0^{(i)}, s_1^{(i)}, \ldots\}_{i=1}^{m} \tag{6.1}$$

As defined by Abbeel and Ng (77), IRL assumes that a k-dimensional feature vector $\phi(s) \in [0,1]$ exists in the state space S of the problem such that $R(s) = w \cdot \phi(s)$ is true. Given a set of states with assigned reward values and a feature vector $\phi(s)$, an optimization step can be introduced to find the weight vector $w \in \mathbb{R}^k[0,1]$ that follows this equation. The feature vector $\phi(s)$ is usually hand-crafted, although some entropy-based approaches have been proposed as automatic selection methods (78).

Given a Markov Decision Process, where the reward function $R(s)$ is unknown, the goal of the IRL algorithm is to find the optimal policy π^*, defined by the reward function $R(s)$, that satisfies Eq. 6.2:

$$||\mu(\pi^*) - \hat{\mu}_E||_2 < \varepsilon \tag{6.2}$$

where $\mu(\pi^*)$ is defined following Eq. 6.3:

$$\mu(\pi^*) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \middle| \pi^*\right] \in \mathbb{R}^k \tag{6.3}$$

and $\hat{\mu}_E$ is the expert feature expectation defined as in Eq. 6.4:

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)}) \tag{6.4}$$

This expert feature expectation parameter encodes which features $\phi(s)$ are preferred by the demonstrator. Desired features are assigned with higher rewards.

Following these premises, different approaches have been proposed to solve the IRL problem. For the NPST framework, a Maximum Entropy IRL approach (79) is used as the IRL algorithm to define the input actions. In this algorithm, the IRL problem is reduced to the maximization of the likelihood distribution defined in Eq. 6.5:

$$\mathscr{L}(w) = logP(E, w|R) = \mathscr{L}_E + \mathscr{L}_w \tag{6.5}$$

where $\mathscr{L}_E$ and $\mathscr{L}_w$ are defined as in the following equations:

$$\mathscr{L}_E = logP(E|R) \tag{6.6}$$

$$\mathscr{L}_w = logP(w) \tag{6.7}$$

The Eq. 6.5 was later adapted by Wulfmeier et al. (78) to work with deep neural networks. This adaptation was achieved by defining the gradient of the reward function as a function of the deep neural network weights obtained via backpropagation. The resulting formula is depicted in Eq. 6.9 extracted from Eq. 6.8.

$$\frac{\delta \mathscr{L}}{\delta w} = \frac{\delta \mathscr{L}_E}{\delta w} + \frac{\delta \mathscr{L}_w}{\delta w} \tag{6.8}$$

where $\frac{\delta \mathscr{L}_E}{\delta w}$ is defined as:

$$\frac{\delta \mathscr{L}_E}{\delta w} = \frac{\delta \mathscr{L}_E}{\delta R} \cdot \frac{\delta R}{\delta w} = (\hat{\mu}_E - \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} P(s_t^{(i)} | R)) \cdot \frac{\delta R}{\delta w} \tag{6.9}$$

Here, as shown by Ziebart et al. (79) the gradient of $\mathscr{L}_E$ with respect the model weights of a linear function is equal to the difference in feature expectation along the expert trajectories.

6.2 Neural Policy Style Transfer

Let Content $\mathbb{C}$ and Style $\mathbb{S}$ be two different DQN trained via IRL corresponding to two different actions defined by two different sets of demonstrations. The same deep neural network architecture –in terms of layers and activation functions, but not weights– can be used for the two networks. Two control policies π_c and π_s can be defined using these DQN.

A control policy π_g can also be defined for this action using a DQN with the same deep neural network architecture of $\mathbb{C}$ and $\mathbb{S}$; let $\mathbb{G}$ be this DQN that encodes the Generated action defined with the execution of the NPST. An architecture similar to the one proposed by Mnih et al. (2) is used for each of the three DQN. The goal of the Generated action encoded by $\mathbb{G}$ can be defined using the goal encoded in the Content action, while the Style action defines the “mood” or “emotion” secondary task in which the Generated action will be performed. The goal encoded by $\mathbb{C}$ is defined as the Content and the secondary task encoded by $\mathbb{S}$ as the Style. The NPST algorithm is proposed to transfer the Content of $\mathbb{C}$ and the Style of $\mathbb{S}$ to $\mathbb{G}$.

The NPST framework is depicted in Fig. 21. The Content transfer step is defined as a backpropagation step over $\mathbb{G}$ with the high-level features of $\mathbb{C}$ as the true labels. These high-level features correspond to the Q-value outputs of the $\mathbb{C}$ network. The obtained backpropagation loss is used to define the Content loss $\mathscr{L}_{content}$. For a single output, this Content loss is defined as the Mean Squared Error of the Q-values as in Eq. 6.10.

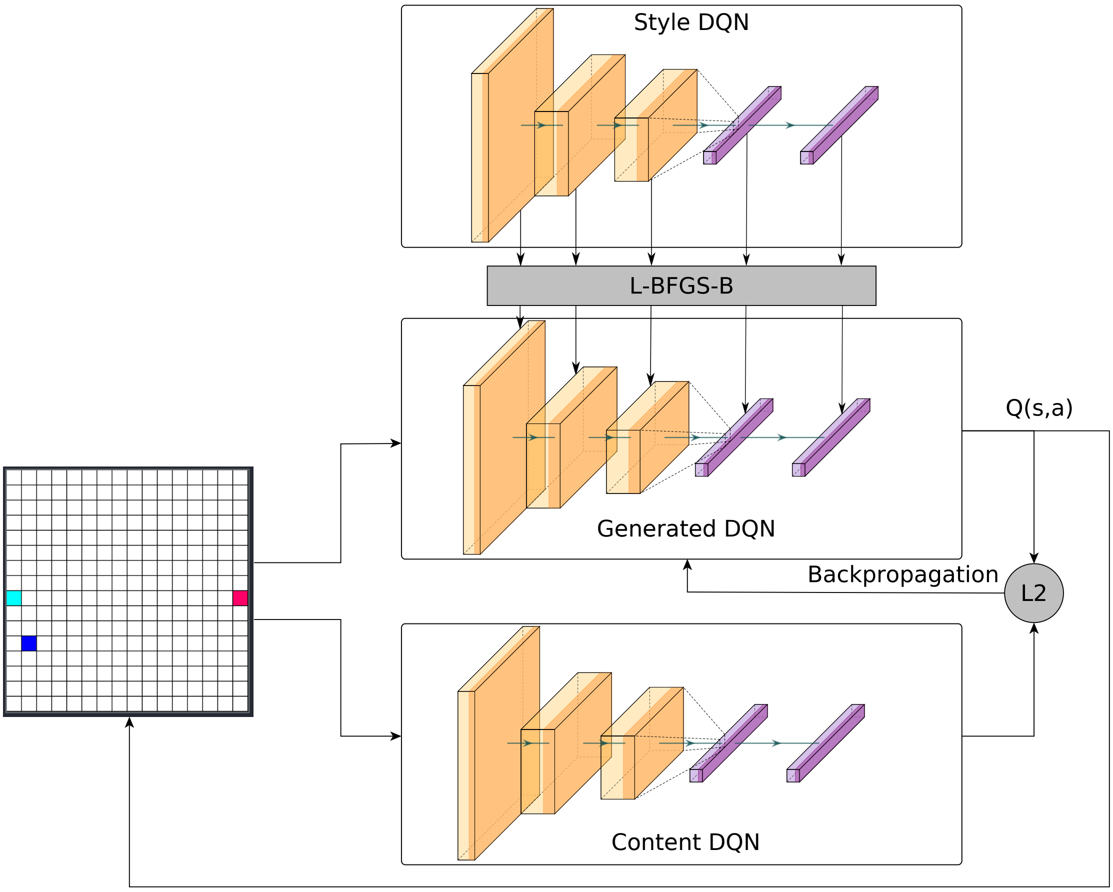

Figure 21: Neural Policy Style Transfer (NPST) framework. Three DQN are proposed with the same architecture. The content DQN ($\mathbb{C}$) is trained using the reward function extracted from the content action demonstrations. The Style DQN ($\mathbb{S}$) is trained using the reward function extracted from the Style action demonstrations. The Generated DQN ($\mathbb{G}$) is obtained using the output layer Q-values of $\mathbb{C}$ and the full weights of $\mathbb{S}$.

$$\mathscr{L}_{content}(\mathbb{G},\mathbb{C}) = ||q_{\mathbb{G}} - q_{\mathbb{C}}||_2 \tag{6.10}$$

where $q_{\mathbb{G}}$ and $q_{\mathbb{C}}$ are the Q-values of $\mathbb{G}$ and $\mathbb{C}$ respectively.

The Style Transfer step is implemented via an optimization step using the weights of $\mathbb{S}$. The Style loss is defined following Eq. 6.11:

$$\mathscr{L}_{style}(\mathbb{G},\mathbb{S}) = ||w_{\mathbb{G}} - w_{\mathbb{S}}||_2 \tag{6.11}$$

where $w_{\mathbb{G}}$ and $w_{\mathbb{S}}$ correspond to the weights of $\mathbb{G}$ and $\mathbb{S}$ respectively.

The full algorithm is shown in Algorithm 8. The intuition behind NPST is that the Content main objective is enforced via successive training specifically with respect the output layer (the Q-values) of $\mathbb{C}$, while the Style is introduced as a secondary task through successive updating via the full weights of the $\mathbb{S}$ network.

Algorithm 8 Neural Policy Style Transfer (NPST)

```
procedure NPST($\mathbb{C}$,$\mathbb{S}$, env, N)

    Initialize:
    $\mathbb{G} \leftarrow \mathbb{S}$
    $w_{\mathbb{S}} \leftarrow \mathbb{S}.get\_weights()$
    $env.init()$
    $state \leftarrow env.observe()$

    for n=1:N do
        Update Environment:
        $q_{\mathbb{G}} \leftarrow \mathbb{G}.predict(state)$
        $a_{\mathbb{G}} \leftarrow argmax_a(q_{\mathbb{G}})$
        $env.step(a_{\mathbb{G}})$
        $state \leftarrow env.observe()$

        Content Transfer:
        $q_{\mathbb{C}} \leftarrow \mathbb{C}.predict(state)$
        $q_{\mathbb{G}} \leftarrow \mathbb{G}.predict(state)$
        $\mathbb{G}.backprop(q_{\mathbb{G}}, q_{\mathbb{C}})$

        Style Transfer:
        $w_{\mathbb{G}} \leftarrow \mathbb{G}.get\_weights()$
        $w_{\mathbb{G}} \leftarrow \text{L-BFGS-B}(||w_{\mathbb{G}} - w_{\mathbb{S}}||_2)$
        $\mathbb{G}.set\_weights(w_{\mathbb{G}})$
    end for


end procedure
```

Here, $\mathbb{G}$ is initialized with the weights of $\mathbb{S}$. Before the execution loop, the simulated environment is initialized and the state is observed. The network $\mathbb{G}$ is updated for N iterations. The loop is divided in the following steps:

1. The output of $\mathbb{G}$ is used to update the environment.
2. The Q-values $q_{\mathbb{C}}$ and $q_{\mathbb{G}}$ of $\mathbb{C}$ and $\mathbb{G}$ respectively are obtained. A backpropagation step is performed over $\mathbb{G}$ using these Q-values.
3. The Style Transfer step is defined using a box-constrained limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS-B) algorithm (80) to update the weights of $\mathbb{G}$. This optimization step is performed following Eq. 6.11.

In the following sections, two sets of experiments, as proposed in the introduction of this chapter, are presented using this algorithm.

6.3 Experiments

Two different sets of experiments were performed in this chapter to measure the performance of the NPST algorithm. The first set of experiments, involving the Catch-ball game scenario, was inspired by classical DRL experiments introducing Atari games. This Catch-ball game scenario was performed in simulation. For the second set of experiments, the execution of the Grid-world paint action was proposed. This action is similar to the “paint” action presented in previous chapters. Real world executions using TEO as the real humanoid robotic platform are performed in this set of experiments.

6.3.1 Catch-ball Game Experiment

The Catch-ball game scenario is inspired by classical DRL experiments introducing Atari games. This scenario is similar to the Pong arcade game. A ball is released from the top of the screen from a random location. The ball then starts to fall following a vertical line. If the ball touches the ground, it counts as a loss. If the paddle catches the ball before it touches the ground, it counts as a win. The agent can move the paddle horizontally at the bottom of the screen.

Three sets of demonstrations, including five expert demonstrations each, are performed for these experiments. These three sets define the three different actions introduced: one Content action, and two different Style actions ("nervous" and "fall"). The Content action tries to win the game by catching the ball with the paddle. The "nervous" Style imitates a nervous behavior with the paddle. Fast small movements around the same position are performed in the demonstrations of this action. The "fall" Style imitates a fall movement in which the paddle falls to the left of the screen. In the demonstrations of this action, the paddle is constantly moving to the left positions. Both Style actions ignore the position of the ball.

The Maximum Entropy Deep IRL algorithm presented in section 6.1 is introduced to find the reward functions defining these three actions. These rewards functions are later used to train the DQN introduced in the NPST algorithm. In order to use the IRL algorithm, a hand-crafted feature vector $\phi(s)$ and a latent state space s have to be defined for each set of demonstrations. For the Content demonstrations, s is defined as the state space of ball and paddle positions and $\phi(s)$ defines if the paddle and the ball are aligned. For the Style demonstrations, the same latent state space s was introduced for both sets of demonstrations. This state space s is defined as the last three paddle positions corresponding to the last three time steps. The feature vector $\phi(s)$ encodes spatial-temporal information of the paddle. For the "nervous" Style, $\phi(s)$ encodes if a movement was performed with the same starting and ending position. For the "fall" Style, $\phi(s)$ encodes if a movement to the left was performed.

In the case of the Content demonstrations, 5 iterations of the IRL algorithm are required for convergence. This number was reduced to 2 for the Style demonstrations. The IRL discount factor (γ) was set to 0.9 with a learning rate of 0.01 for all sets of demonstrations. The three resulting reward functions $R(s)$ are then used to train the three DQN corresponding to the three actions proposed.

The DQN are defined using the same architecture as the one proposed by Mnih et al. (2). The weights of the networks are initialized with random values. The inputs are raw 80x80 pixel images corresponding to the game screen. The outputs are the Q-values assigned for each of the three possible actions in the game (move paddle to the left, to the right and stay still). The DQN are trained using a Q-learning algorithm.

Table 11: Hyper-parameters defined for the Catch-ball game.

Hyper-parameter	Setting
Shared	
Input size	80x80
Number of input time steps	4
Optimizer	Adam (81)
Loss function	Mean Squared Error
Number of actions	3
Discount (γ)	0.99
Experience Replay size	5000
Q-learning	
Learning Rate	1e-6
Initial Epsilon	0.1
Final Epsilon	1e-5
Epsilon gradient	Lineal
Batch size	32
Exploration Epochs	100
Training Epochs	1000
NPST Algorithm	
Learning Rate	0.01
Number of iterations (N)	One full Catch-ball episode
Batch size	100
L-BFGS-B internal iterations	1

The hyper-parameters chosen for training the DQN and the execution of the NPST algorithms are depicted in Table 11.

6.3.2 Grid-world Paint Experiment

The goal of the Grid-world paint scenario is to reach a target pixel defined within a 16x16 grid environment with the agent. The agent in this scenario is the end-effector of the humanoid robot TEO. The action space allows the agent to move within the Grid-world environment one grid step at a time. Only vertical and horizontal movements are allowed. The grid is physically represented by a large monitor in front of the robot. An auxiliary depth sensor is in charge of measuring the position of the end-effector and updating the grid according to the agent by changing the pixel colors. An image of the final setup, used in this scenario for the IRL training, is depicted in Fig. 20.

Three sets of demonstrations, of five expert demonstrations each, are introduced in this set of experiments. The demonstrations define three different actions similar to the ones used in the Catch-ball experiments: one Content action, and two different Style actions (“nervous” and “fall”). The Content action aims to win the game by reaching the target pixel. The “nervous” Style imitates a nervous behavior or mood by performing small movements around the initial positions. The “fall” Style imitates a fall behavior in which the agent always tend to move towards the bottom side of the grid environment. Both Styles ignore the target pixel.

As in the Catch-ball game experiments, the Maximum Entropy Deep IRL algorithm presented in section 6.1 is introduced to find the reward functions that defines these three actions. These rewards functions are then used to train the DQN defined within the NPST algorithm. A hand-crafted feature vector $\phi(s)$ and a latent state space s are defined for each set of demonstrations. For the Content demonstrations, the state space s is defined as a 256 element vector resulting from performing a flatten operation over the 16x16 pixel grid. The feature vector $\phi(s)$ encodes if the current position of the agent corresponds to the target pixel. The same state space s

used for the Content demonstrations is used for the “fall” demonstrations. For these demonstrations, the feature vector $\phi(s)$ encodes if the current agent position corresponds to a position at the bottom of the grid. For the “nervous” demonstrations, the state space s is a 16^3 element vector, where 16 is the number of possible vertical positions and 3 corresponds to the last three time steps. The feature vector $\phi(s)$ for this Style encodes if the a movement was performed with the same starting and ending position.

Table 12: Hyper-parameters defined for the Grid-world paint scenario.

Hyper-parameter	Setting
Shared	
Input size	16x16
Number of input time steps	4
Optimizer	Adam (81)
Loss function	Mean Squared Error
Number of actions	4
Discount (γ)	0.99
Experience Replay size	50000
Q-learning	
Learning Rate	1e-6
Initial Epsilon	0.9
Final Epsilon	0.01
Epsilon gradient	Lineal
Batch size	32
Exploration Epochs	100
Training Epochs	5000
NPST Algorithm	
Learning Rate	0.01
Number of iterations (N)	One full Grid-world episode
Batch size	100
L-BFGS-B internal iterations	10

For each of the three sets of demonstrations, 5 iterations of the IRL algorithm were required. The discount factor (γ) was set to 0.9 and a learning rate of 0.01 was introduced. The three rewards functions $R(s)$ obtained were used to train the DQN defining the three actions. The same architecture for the DQN as the one used in the Catch-ball game is used in these experiments. The DQN are initialized with random weights values. The inputs of the networks are the 16x16 matrix representing the state of the grid world. The outputs are the Q-values assigned for each of the four possible actions (up, down, left, right). Q-learning is the training algorithm used to train the networks. The hyper-parameters chosen for training the DQN and the execution of the NPST algorithms are depicted in Table 12.

6.4 Results

To measure the performance of the NPST algorithm, three parameters were measured in the experiments: the Content loss $\mathscr{L}_{content}$, the Style loss $\mathscr{L}_{style}$, and the win ratio. The results depicted are the average results obtained from 10 executions of the NPST algorithm. In order to have a baseline to easily understand the obtained results, the performance of the three vanilla DQN defining the three base actions were also added for the two sets of experiments.

Table 13: Catch-ball game experimental results.

Actions	$\mathscr{L}_{style}$	$\mathscr{L}_{content}$	$Wins(\%)$
Content[1]	16796.47 16809.26	—	100
Nervous Style	—	1.33	0
Fall Style	—	0.50	0
Nervous NPST	62.21	0.81	70
Fall NPST	47.91	0.39	80

6.4.1 Catch-ball Game Results

For all the repetitions of the Catch-ball experiments, the ball was released from the same initial position at the top of the screen. These experiments were performed in simulation. The obtained results are depicted in Table 13. Here, the "Nervous Style" and the "Fall Style" action correspond to the base Style actions defined by the demonstrations. These results illustrate effects such as the NPST "fall" Style obtaining a lower $\mathscr{L}_{content}$ than the NPST "nervous" Style. This effect can be related to the initial position of the ball being on the left side of the screen. This is also the side where the preferred states for the "fall" Style are. This makes both the Content and the "fall" Style policies to search for the same states. The results show a reduction in both $\mathscr{L}_{content}$ and $\mathscr{L}_{style}$ for both obtained policies with respect the vanilla DQN.

---

[1] For the Content action, two different Style losses ($\mathscr{L}style$) were measured. The value at the top corresponds to the loss with respect the "nervous" Style, while the value at the bottom is with respect the "fall" Style.

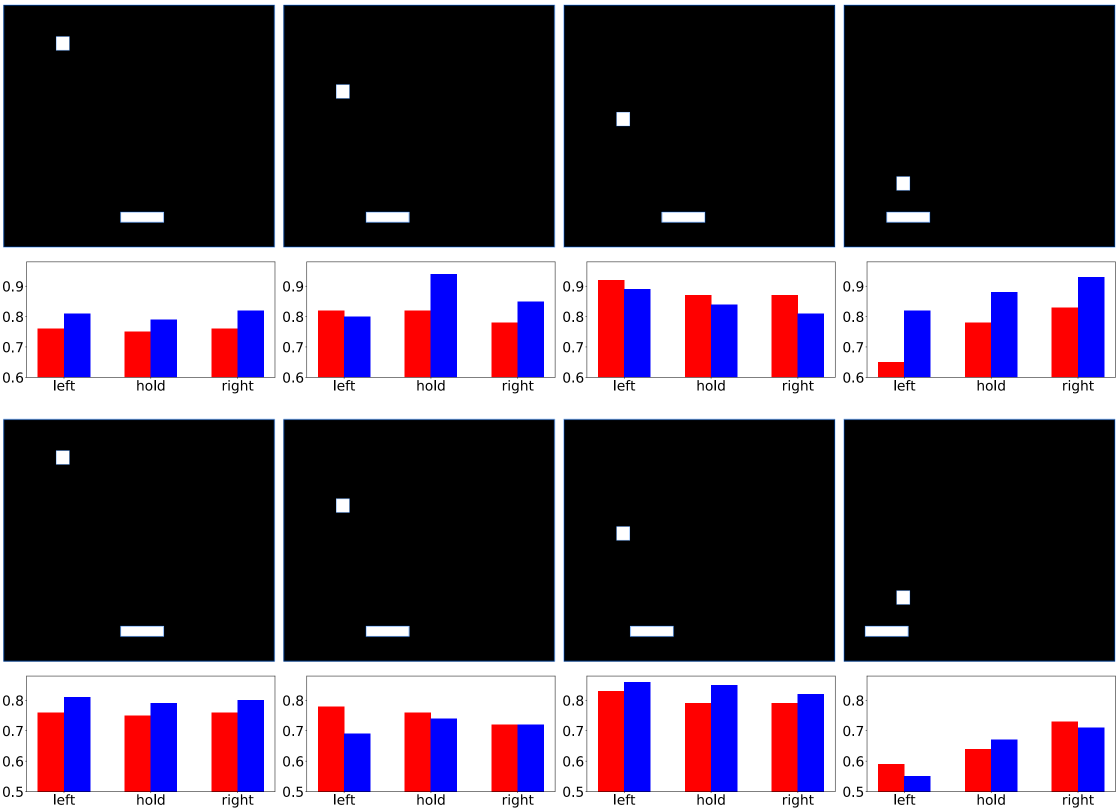

Figure 22: Screenshots obtained during the NPST execution with the "nervous" Style (top) and "fall" Style (bottom). Red bars are the output of the Content network $\mathbb{C}$. Blue bars are the output of the Generated policy $\mathbb{G}$.

Fig. 22 depicts the state of intermediate time steps defined by the NPST Generated policy. The intermediate steps were randomly selected with the condition of being equidistant and the same for both Styles. The first and third row depict screenshots of the game state. These game states were generated following the two Generated policies corresponding to the two

Styles. The bar graphs compare the Q-values obtained with the Content policy and the Generated policies.

6.4.2 Grid-world Paint Results

In addition to the three parameters explained in the introduction of this section, a new parameter, the number of steps, was also introduced for this set of experiments.

Table 14: Grid-world paint experimental results.

Actions	$\mathscr{L}_{style}$	$\mathscr{L}_{content}$	$Steps$	$Wins(\%)$
Content[2]	1085.91 1055.98	—	16.0	100
Nervous Style	—	3.02	50	0
Fall Style	—	10.87	50	0
Nervous NPST	715.29	0.07	43.6	100
Fall NPST	336.75	1.30	34.6	40

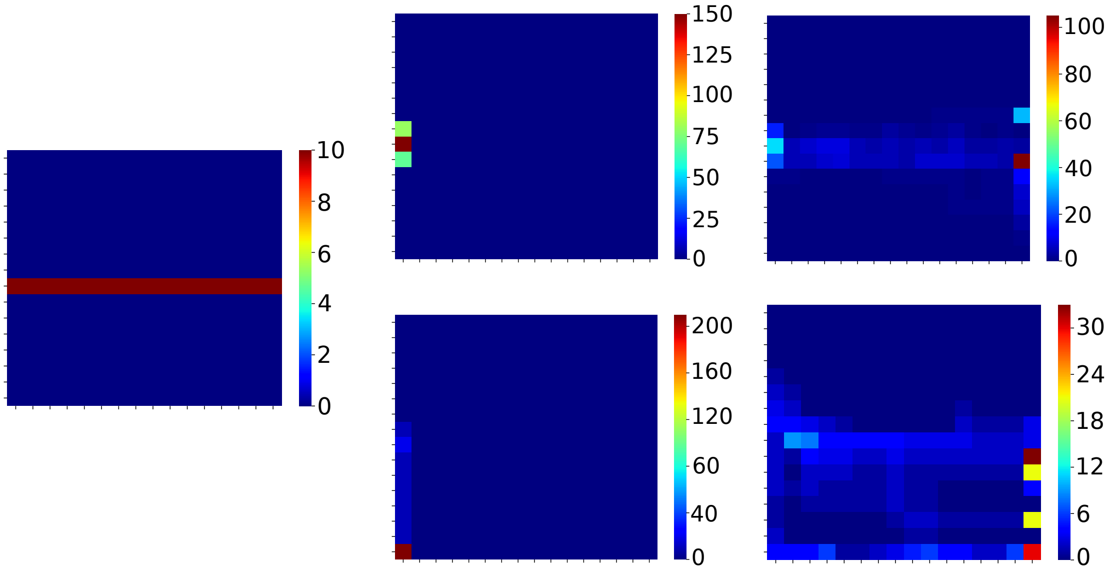

Figure 23: Agent heat-map. From left to right: Content, Style and Generated policies. "Nervous" Style (top) and "fall" Style (bottom). Pixels with warms colors have been recurrently visited by the agent. Cold colors represent less visited pixels. The depicted scale is the cumulative result of the 10 different executions of the NPST algorithm.

---

[2] As in the Catch-ball game experiments, the value at the top corresponds to the loss with respect the "nervous" Style, while the value at the bottom is with respect the "fall" Style.

The obtained results are depicted in Table 14. The results depict a pronounced difference between the $\mathscr{L}_{content}$ obtained with each of the Styles. This difference impacts the performance of the Generated policies. As observed in the results of the Catch-ball game environment, lower $\mathscr{L}_{content}$ comes with an increase of the performance. In this scenario, the "nervous" Style reaches the same ratio of wins as the Content policy. One hypothesis behind this is that the "nervous" Style allows the Generated policy to follow an approximately straight line similar to the one used by the Content. The results also show how, although the "nervous" Generated policy reaches the same performance as the Content, the number of required steps is increased.

The paths performed by the different policies can be observed in Fig. 23. This figure encodes the most and less visited states by the different agents. The first and second columns of the figure correspond to the vanilla DQN. The third column are the obtained Generated policies using NPST. The depicted scale are the cumulative results of 10 repetitions. Warm colors depict recurrently visited pixels, cold colors depict less visited pixels. Both Generated policies try to reach the target pixel while introducing the Style. In the "nervous" Style, the most visited pixels are the ones defined by the target pixel and his adjacent vertical neighbors. For the "fall" Style, the most visited pixels are the one between the target pixel and the bottom of the screen.

6.5 Conclusions

In this chapter, the NPST algorithm has been proposed as a way to perform Style Transfer between control policies. The goal of Style Transfer is to introduce some Style in a certain input without changing the original Content. NPST allows the adaptation of DRL control policies to different contexts through the introduction of different Styles.

In NPST, actions are defined using DQN. Three DQN are defined within the NPST framework: the Content DQN ($\mathbb{C}$) containing the Content; the Style DQN ($\mathbb{S}$) containing the Style; and the Generated DQN ($\mathbb{G}$) which is the output of the NPST algorithm. A Maximum Entropy Deep IRL is proposed to define the Content and Style DQN via user demonstrations. The Content loss is defined as the difference between the outputs of $\mathbb{C}$ and $\mathbb{G}$. The Style loss is defined as the difference in the weights of $\mathbb{S}$ and $\mathbb{G}$. The Generated policy is the output of the NPST algorithm.

Two different sets of experiments corresponding to two different actions were performed in this chapter: the Catch-ball game action and the Grid-world paint action. The first action is inspired on classical DRL problems using Atari games. The second action, Grid-world paint, is similar to the "paint" action presented in previous chapters. Two different Styles ("nervous" and "fall") were introduced in the experiments. The results show a clear influence of the transferred Style. The generated control policies are a successful combination of the Content goal and Style emotion.

Some limitations are present in the framework. The first one is that this framework is only valid for discrete action spaces, while robotic environments are complex environments that cannot be easily discretized. The second limitation comes from how the Content and Style losses are defined. In the experiments, it has been observed that the NPST algorithm converges to solutions similar to the Content. The number of NPST iterations has to be tuned to avoid this. In the following chapter, a new framework to solve these two problems is proposed. This new framework introduces state of the art DRL techniques for working in continuous actions spaces, and the introduction of Autoencoders to extract the Content and Style.

7. NPST3 IN CONTINUOUS ACTION SPACES

Robotic applications are typically characterized by dynamic and complex environments with a low tolerance to error. The introduction of continuous actions spaces for the execution of robot trajectories is usually required to work with these environments. In this chapter, the Neural Policy Style Transfer TD3 (NPST3) framework is proposed. This framework is presented as an advanced version of the NPST framework proposed in the previous chapter. In NPST3, the Content and Style are defined using Autoencoders. The generation of the control policies is performed by a TD3 algorithm. This algorithm allows the robot to work with continuous action spaces. The output of the TD3 network defines the Cartesian velocity of the end-effector to perform the trajectory that minimizes the Content and Style loss. Additionally, the proposed NPST3 framework is designed to allow the introduction of partially generated actions. This allows the online generation of the input action, as in the case of teleoperating the robot while performing the Style Transfer step at the same time. The framework base idea is depicted in Fig. 24.

Different Styles can be defined using different sets of demonstrations. Four emotions are defined as the four selected Styles for the experiments: Angry, Happy, Calm and Sad. Four sets of demonstrations following the four proposed Styles are performed by a volunteer and recorded using a mo-cap system. Each of these Styles are transferred to the same base Content action using the NPST3 framework. The resulting action is executed with a Panda robot (manufactured by Franka Emika GmbH). The results are evaluated using a questionnaire with 73 volunteers asked to assign each unkown Style to each of the resulting actions.

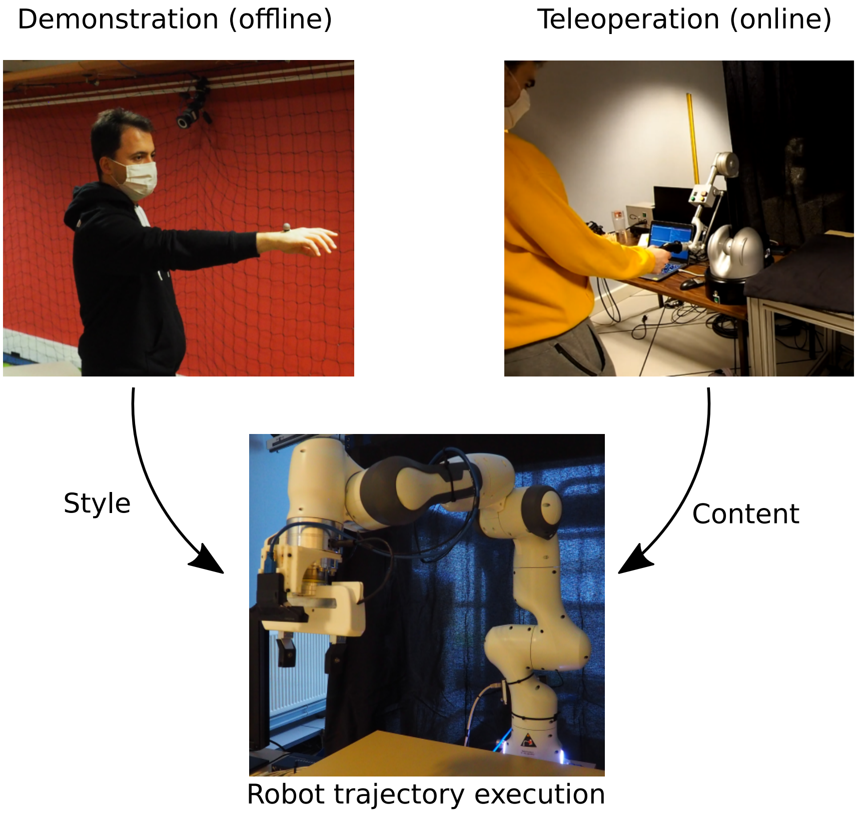

Figure 24: NPST3 base idea: human demonstrations define the Style while the Content can be defined online via teleoperation.

7.1 Framework

Two main stages are introduced to define the NPST3 framework: the Style Transfer stage extracts the Content and Style of the input actions to generates the loss, while the DRL stage is in charge of generating the control policies that produce the actions that minimize this loss. Two neural networks corresponding to these two stages are defined: the loss and the execution network. The loss network is defined within the Style Transfer stage, while the execution network is defined in the DRL stage. The final framework structure is depicted in Fig. 25. The details and architecture of the NPST3 framework is presented in the following sections.

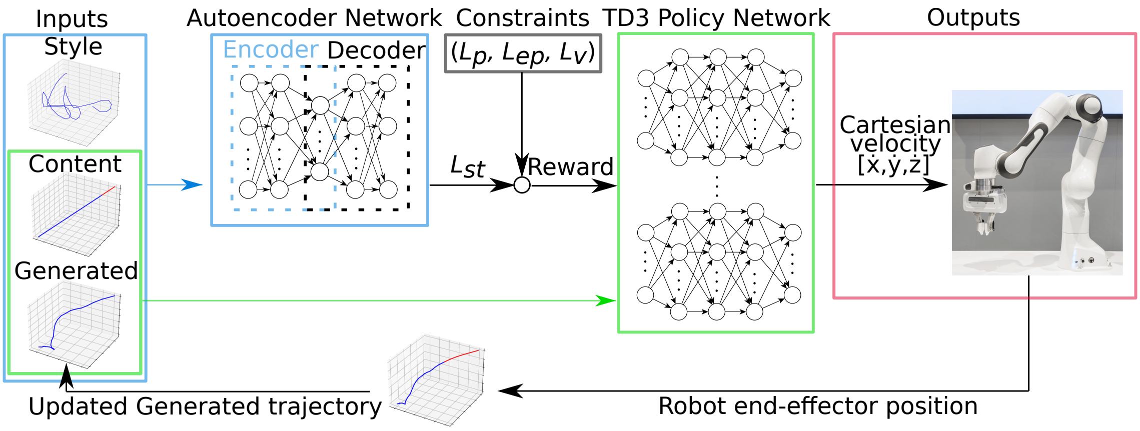
Inputs
Style
Content
Generated
Autoencoder Network
Encoder
Decoder
Constraints
(Lp, Lep, Lv)
L_{st}
Reward
TD3 Policy Network
Outputs
Cartesian velocity
[ẋ,ẏ,ż]
Updated Generated trajectory
Robot end-effector position

Figure 25: NPST3 framework: The input of the Autoencoder network are the Style, Content and Generated actions. The loss obtained with the Autoencoder (L_{st}) is added to the loss defined by the Constraints (L_p, L_{ep}, L_v) to obtain the total loss L. The inverse of this loss is used as the reward for training the TD3 Policy Network. In addition to this reward, the input of the TD3 network are the Content and Generated actions. This network defines a control policy that outputs a 3D Cartesian velocity to the robot end-effector. The velocity command is executed and the new position of the robot is obtained. The Generated trajectory is updated using this new position. The Content trajectory is updated using the information provided by the user. This Content trajectory can be defined offline or online via teleoperation.

7.1.1 Inputs

The inputs of the framework are three 3D Cartesian trajectories (x,y,z) defining the Content, the Style and the Generated actions. The length of all the input trajectories is limited to 5 seconds. The sample rate is set to 10 samples per second to compromise accuracy and computational cost. Data with a higher sample rate is reduced to this ratio. Input trajectories are defined using an $[m,n]$ vector, where m is set to 50 corresponding to the total number of samples, and n is set to 3 corresponding to the total number of dimensions. This framework allows introducing incomplete trajectories as inputs, which allows the online generation of the Generated and Content actions. This is the case for example of teleoperating the robot while the NPST3 framework is being executed. Incomplete actions are padded with zeroes to have the shape required by the network.

7.1.2 Autoencoder Network: loss network

The loss network is defined using a Convolutional Autoencoder. A similar architecture to the one proposed by Holden et al. (12) is introduced to define this Autoencoder network. The encoder is defined using a 1D convolutional layer followed by a pooling operation. This Convolutional layer is composed by 256 filters with a kernel size of 5. Two layers equivalent to the transposed layers of the Encoder are introduced to define the Decoder. An additional dropout layer is defined in the Encoder to improve the performance of the Autoencoder network. The resulting loss network extracts the Content and Style of the input actions to define the loss in Eq. 5.5.

7.1.3 Constraints

In addition to the loss defined by the loss network, some additional constraints are introduced to assure that the generated motion is achievable by the robot. Following this idea, the Generated trajectory should be similar to the one defined by the operator: there should not be big jumps within the execution of the full trajectory, and there should be some control over the obtained trajectory velocity. These constraints are defined within the position and velocity spaces of the robot.

The first constraint limits the position error of the Generated action with respect to the Content action. A loss following Eq. 7.1 is introduced each step.

$$L_p = \|\frac{G[t-1] - C[t-1]}{RT}\| \tag{7.1}$$

where t is the current execution step and RT is the action Robot Threshold (RT) defined by the robot.

The second constraint is introduced to limit the last step position error of the Generated trajectory with respect the Content trajectory. This constraint smooths the transition between multiple Generated consecutive actions. This constraint is defined as in Eq. 7.2.

$$L_{ep} = \| \frac{G[t_n] - C[t_n]}{RT} \| \tag{7.2}$$

where t_n is the last time step. If the last time step has not been reached yet, this loss is set to 0.

A final constraint is introduced in the velocity space of the robot. This constraint increases the weight of the velocity transfer within the Style Transfer algorithm. This constraint is introduced following Eq. 7.3.

$$L_v = \| (\frac{\partial G}{\partial t} - \frac{\partial S}{\partial t})/RT \| \tag{7.3}$$

The total loss to train the execution network is then defined as the weighted sum of the losses defined by these constraints and the Content and Style losses defined by the Autoencoder network. This total loss is depicted in Eq. 7.4.

$$L = w_c L_{content} + w_s L_{style} + w_p L_p + w_{ep} L_{ep} + w_v L_v \tag{7.4}$$

7.1.4 TD3 Policy Network: execution network

The execution network is defined by a TD3 network that generates the control policies that produce the Generated action which minimizes the total loss defined in the Style Transfer stage. The Content and Generated actions are the inputs of this network. Both actions are incrementally introduced to the network. This allows the online introduction of the Content action. The Style is implicitly encoded in the network and defined during training. One TD3 network has to be trained for each Style. The same network can be used for any Content action.

Two different networks are defined within the TD3 architecture: the actor network and the critic network. The control policy is encoded by the actor while the Q-value function is encoded by the critic. For both networks the two input actions (Content and Generated) are used as input. For the actor network, first, each input passes through three 1D convolutional layers separately. The first convolutional layer is defined using 256 filters with a kernel size of 5. This architecture was chosen to be the same as the layer introduced for the Encoder. The rest of the convolutional layers each have 128 filters with a kernel size of 5. The two outputs of the convolutional layers are flattened and concatenated. The resulting concatenated output is passed to four fully connected layers. The first two fully connected layers have a total of 512 nodes, the third 400, and the last one 300. Between each of the layers of the network, a batch normalization layer is introduced. A Rectified Linear Unit (ReLU) (82) activation was chosen for all the layers except the last one, which was defined using a hyperbolic tangent (tanh) activation. The output of the network is a 3D vector with the Cartesian velocity of the robot end-effector.

The critic network is designed with the same architecture as the actor network with some exceptions: the TD3 action, defined as the output of the actor network, is introduced as an additional input; the TD3 action input is not passed through the convolutional layers, but instead it goes directly to the fully connected layers; the TD3 action is concatenated with the rest of the inputs after the first fully connected layer; an additional layer with 512 nodes is added before the 400 node layer; the last layer of the network uses a linear activation function instead of a hyperbolic tangent (tanh) activation; and the output of the critic network is a single floating-point value corresponding to the Q-value.

7.1.5 Outputs

The output of the framework is a 3D Cartesian velocity vector that can be used to control the end effector of the robotic platform. The new robot position is used to update the Generated action trajectory. This Generated action trajectory is then used as input for the next step of the framework.

7.2 Training

In a similar way to the one proposed by Li et al. (83), the loss network was trained using the CMU Graphics Lab Motion Capture Database[1]. The CMU database contains recordings of different motions performed by different individuals using a mo-cap system. Only the data corresponding to the tracker at the end of the right hand (RFIN) was introduced. Using this dataset, the Autoencoder defining the loss network was trained to produce the same output trajectory as the selected input trajectory.

For training the execution network, a random generator of linear trajectories was implemented as the Content action generator. In the case of the Style actions, four different Styles were selected: Angry, Happy, Calm and Sad. The four Styles actions defining these four Styles were defined via human demonstrations. These demonstrations were performed by a volunteer and recorded by a mo-cap system. The only instruction given to the volunteer was to move expressing the given emotion. Only the position of the right hand of the volunteer was recorded during these demonstrations. One cut of 5 seconds long from each of the four Style motions performed by the volunteer was selected for training. The results of the training were four execution networks, each encoding one of the four selected Styles. The TD3 algorithm was the selected algorithm for training. The loss defined in Eq. 7.4 was inverted and introduced to the TD3 algorithm as the reward. The training parameters of the execution and loss networks are depicted in Table 15.

---

[1] http://mo-cap.cs.cmu.edu/

Table 15: Training hyper-parameters for the NPST3 algorithm.

Hyper-parameter	Setting Value
Shared	
Input trajectory shape	[50,3]
Sample frequency	10 [Hz]
Trajectory length	5 [s]
Robot Threshold (RT)	300 [mm]
Optimizer	Adam (81)
Number of Styles	4
Autoencoder	
Epochs	1000
Batch size	256
TD3 Network	
Action space dimensions	3
Action Range (AR)	± 0.1 * RT
Loss weights $(w_c, w_s, w_p, w_{ep}, w_v)$	(100, 1, 0.1, 1, 20)
Epochs	2500
Experience Replay size	10e3
Batch size	64
Critic Learning Rate	1e-5
Actor Learning Rate	1e-6
Discount (γ)	0.99
Critic/Actor update ratio	2
Target update value (tau)	1e-3
Initialization network values	± 3e-3 (Uniform distribution)
Loss Function	Mean Squared Error
Policy noise	0.002 * RT (Normal distribution)
Action noise	0.02 * RT (Normal distribution)

7.3 Experiments

To evaluate the performance of the framework, a questionnaire with human volunteers was performed. In this questionnaire, four videos corresponding to the four generated motions resulting of the four selected Styles were shown to the volunteers. The goal of the questionnaire was to find if the volunteers were able to detect the original emotion transferred to the action.

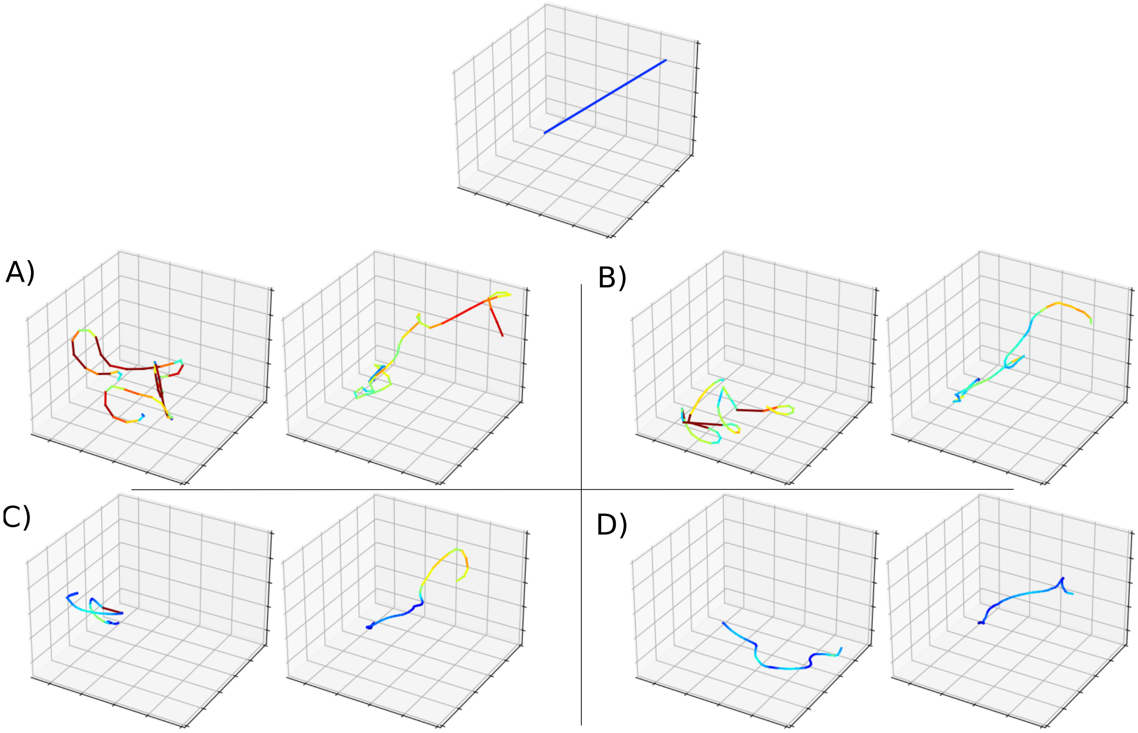

Figure 26: Cartesian trajectories obtained in the experiments. The top trajectory is the selected Content action. For each pair of trajectories, the left trajectory corresponds to one of the four selected Styles: A) Angry, B) Happy, C) Calm and D) Sad. The trajectory in the right corresponds to the generated action obtained with the NPST3 framework. Style trajectories are defined via demonstrations with the right hand of a volunteer using a Vicon mo-cap system. Warm colors depict a higher velocity value, while cold colors depict lower velocities.

7.3.1 Setup

The Content action selected was the one defined by the same Cartesian trajectory as the one depicted at the top of Fig. 26. This Content action was used as input for the four trained execution networks corresponding to the four Styles: Angry, Happy, Calm and Sad. The resulting control policies were executed on a Franka Emika Panda robot. The robot end-effector frame was used as the reference frame for the execution of the output velocities. The orientation of the end-effector was kept the same for the full execution. ViSP (84) was introduced as the control library to perform real time executions with the robot. The Generated results are the 5 seconds long Cartesian trajectories depicted in Fig. 26. The four full executions were recorded using a camera.

7.3.2 Questionnaire and Subjects

The questionnaire was divided in two parts. In the first part, the volunteers were presented with the four videos and asked to freely write the emotion that better suits the robot movements. No additional restrictions were added in this part. In the second part, the four videos were presented with the four emotions as selectable options: Angry, Happy, Calm and Sad. The volunteers were asked to select one emotion for each of the videos. Each emotion could not be selected more than once.

A total of 73 volunteers were asked to complete the questionnaire. These volunteers were selected from different areas of work, academic levels and a wide range of ages (18 to 76 years old). It was not limited to people familiar with robots. The original questionnaire was performed in three different languages with people from three different nationalities: Spanish, Italian and English.

7.3.3 Results

Results corresponding to the first part of the questionnaire are depicted in Fig. 27. All the obtained results were translated to English using the same translator engine. Here, emotions similar to the original transferred emotions were the most selected for each of the videos. The results corresponding to the second part are depicted in Fig. 28. In this second part, the original transferred emotion was successfully chosen by most of the volunteers for each of the presented videos. As expected, in the case of the Angry emotion the second most selected emotion was Happy and vice versa. The same thing happened for the Calm and Sad emotions.

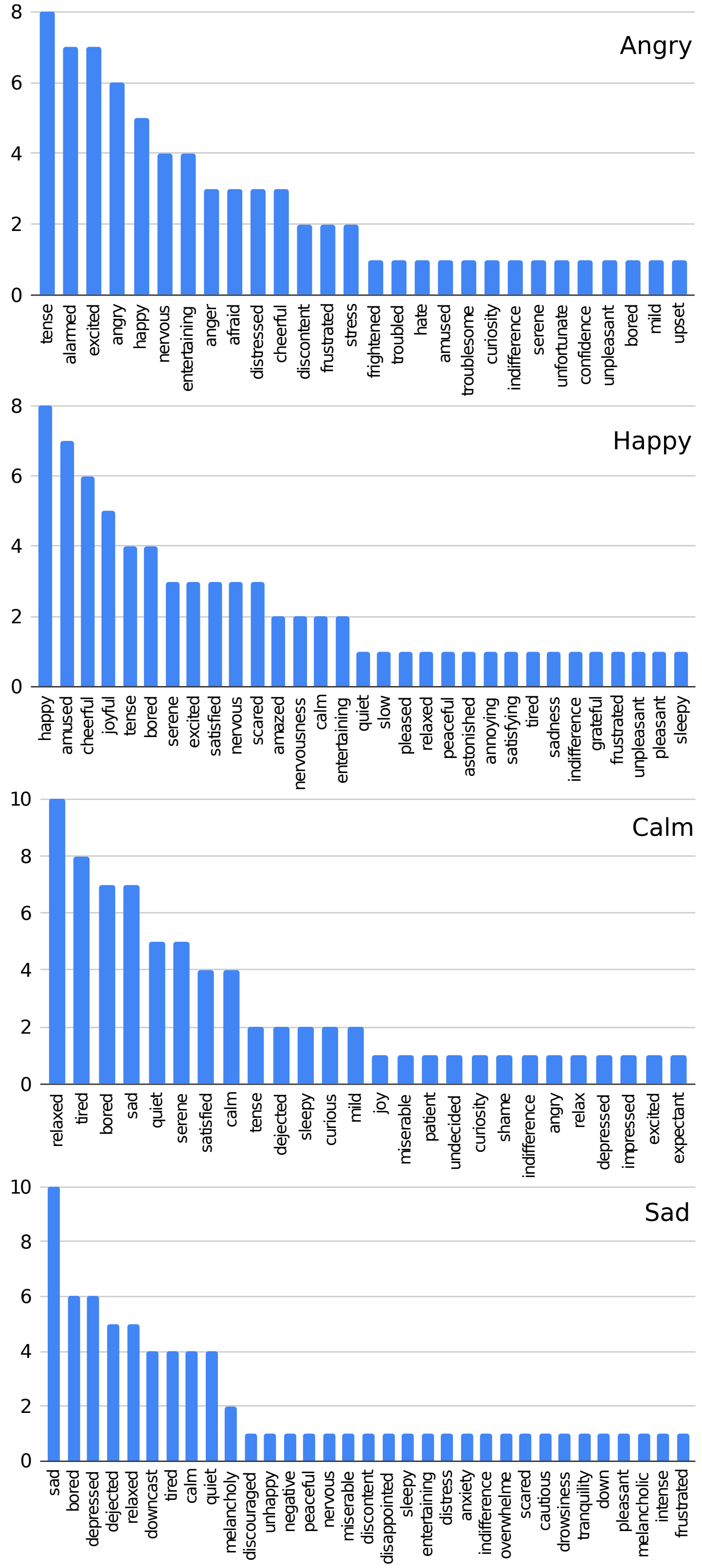

Figure 27: NPST3 questionnaire first part results. Each chart depicts the answers given for one of the videos. The title on the top right of the chart depicts the original emotion transferred to the shown video. The labels on the X axis are the emotions written by the volunteers. The Y axis is the number of times that emotion was used to describe the video by a volunteer.

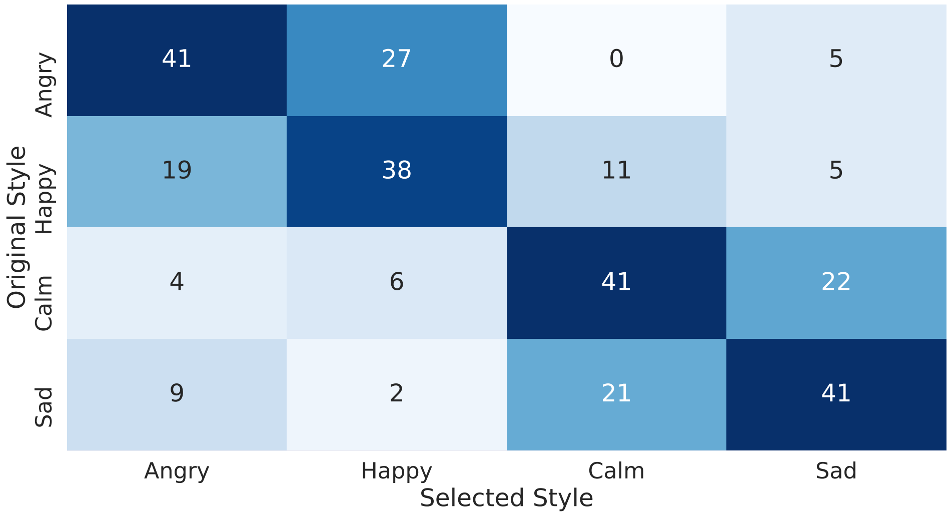

Figure 28: NPST3 questionnaire second part results. Each row corresponds to a different video. Each column corresponds to the Style selected by the volunteers. Each cell contains the number of times a given Style was selected by a volunteer. All the videos were presented to all the volunteers.

7.4 Conclusions

The NPST3 framework has been proposed in this chapter as a way to introduce DRL within an Style Transfer framework. The Content and Style of the base actions are extracted using Autoencoders. New actions are generated using the TD3 algorithm. This algorithm is proposed as a robust robot controller to work in continuous action spaces. The framework is designed to work with incomplete actions. This allows the online generation of the Content and Generated actions. This is the case for example of teleoperating the robot while performing the Style Transfer step. For the experiments, the Franka Emika Panda robot platform was introduced to execute the Generated robot actions. To evaluate the results, the recorded robot executions of the Generated actions are presented to 73 volunteers and asked to identify the original Style. The results depict that the subjects are able to identify the original emotion transferred to the base action.

CONCLUSIONS OF PART III: NEURAL POLICY STYLE TRANSFER

The introduction of Style Transfer within a DRL framework has been proposed within two different approaches in this part of the thesis. The first approach, NPST, was presented as an initial approach using discrete action spaces and IRL. In this first approach, the Style and Content losses were directly defined using the Policy network. This presented the advantage of not needing a secondary network to define these losses. At the same time, this method required some additional previous tuning to properly work. The number of iterations had to be tuned to achieve the desired policy behavior. In addition to this, this network was only designed to work with discrete action spaces.

The second approach presented was the NPST3 framework. This framework improves the base NPST framework in several ways. First, the TD3 algorithm, for the generation of control policies within continuous action spaces, was used in the framework. This allowed the robot to work with continuous actions spaces similar to the ones presented in real robot applications. As a second improvement, an Autoencoder is used to define the Content and Style losses, making the policy training stable without requiring any additional tuning. The final improvement was introduced in the design of the framework. The NPST3 framework was designed to work with incomplete actions. This allows the user to command the robot online while the Style Transfer process is being executed. The result is a robot that is moving as commanded by the user while executing the selected Style.

Results for both approaches show how the Generated control policy is able to perform a new trajectory that introduces the Content and Style defined. For the NPST framework, experiments were performed in two different scenarios. The results are compared in terms of Content and Style loss and action performance. In the case of the NPST3, experiments were approached using a group of volunteers to find if the original Style has been successfully transferred to the Generated motion. Both experiments approach measuring the Style Transfer step with different perspectives. The results obtained depict the generated trajectory as a combination of the Content and Style selected as inputs.

Part IV

Conclusions

8. CONCLUSIONS

This thesis has focused on finding solutions to the Action Generalization problem through diverse theoretical frameworks. Results have been supported by empirical values obtained from the corresponding experimental setups. New and improved frameworks were proposed to increase the Action Generalization capabilities of robots. Experiments have been provided to evaluate the performance of the proposed frameworks, and to compare them with already existing or future related works. In the following sections, the main conclusions extracted from this work are presented, followed by a selection of the main contributions introduced in this work and the future works proposed.

8.1 Main Conclusions and Results

The problem of Action Generalization is currently one of the hardest problems in the robotic community. Two different approaches to increase the Action Generalization capabilities of robots were proposed in this thesis. The first one consisted on the study and improvement of CGDA to work with real robots and environments. In the second approach, NPST, a framework introducing DRL and Style Transfer has been proposed as a way to introduce Styles in robotic actions.

For the first approach, different methods to improve CGDA to work with real world environments were studied. First, the introduction of evolutionary algorithm approximations and constrained genetic algorithms to reduce the number of evaluations was proposed. Evolutionary algorithms approximations modify the definition of evolutionary algorithms to only evaluate part of the population. Constrained genetic algorithms reduce the search space of the robot to achieve a faster convergence. These two methods were tested using the “paint” and “wax” actions proposed in the original CGDA publications.

After studying the experimental results, it was concluded that these algorithms required the introduction of an additionally tuning step. This additional tuning step limited the generalization capabilities of CGDA. In order to deal with this limitation, OET was proposed as an alternative for the execution of CGDA in real world environments. OET is an online evolutionary strategy where real world executions are introduced in the planning stage. This allows the model to be updated online during the execution. To measure the performance of OET, the RIT parameter was introduced to measure the time between environment updates.

In the second approach, NPST was proposed as a way to introduce Style Transfer within a DRL architecture. The idea was to allow the robot to adapt an action to different contexts with the introduction of different Styles. In NPST, the Content and Style of each action are defined via user demonstrations and IRL. Two different sets of experiments were performed. The Catch-Ball game experiments were defined based on classical DRL problems, while the Grid-world paint experiments were similar to the “paint” experiments performed in the first chapters of this thesis.

The NPST framework presented two main limitations: it required the definition of the policy within a discrete action space; and the proposed definition of Content and Style required some additional previous tuning. The NPST3 framework was proposed to solve these limitations. In NPST3, control policies are generated using the TD3 algorithm. This algorithm allows the generation of DRL control policies within continuous action spaces. Additionally, Autoencoders were introduced as a more conceptually accurate way to define the Content and Style. Styles are defined using demonstrations recorded via a mo-cap system.

Several experiments were performed to test the performance of the proposed frameworks. For the first approach, the introduction of approximations and constraints within evolutionary algorithms resulted in relevant reductions in the number of evaluations required for the CGDA framework to converge. In most scenarios, this reduction was not enough to perform all the evaluations directly in real world scenarios. As an alternative, the OET algorithm was proposed as a new evolutionary strategy for the execution of the CGDA framework in dynamic environments. Results for the OET experiments depict reductions in the RIT from minutes to seconds without loosing performance. For the second approach, experiments with the NPST framework depict the generated trajectory as a combination of the Content and Style actions. Here, the Generated trajectory achieved a reduction in the obtained Content and Style losses. The NPST3 framework was proposed as a more advanced version of NPST for working with continuous action spaces. Experiments for the NPST3 algorithm were performed with a group of 73 volunteers. For these experiments, volunteers were presented with a recorded video of the robot executing the NPST3 generated action. The same Content action was used for all the videos. The volunteers were able to successfully identify the original Style transferred to the robot action.

8.2 Innovations

Two different approaches to deal with the problem of action generalization have been proposed in this work. These two approaches introduce two main contributions to the area of Action Generalization in robotics. The first contribution proposes the improvement of CGDA to work with dynamic environments. The second contribution introduces Style Transfer with a Deep Reinforcement Learning controller for its application in robotic actions.

- **The improvement of CGDA to work with dynamic environments**. In this thesis, several methods to improve the CGDA execution step with real world scenarios have been proposed within two different approaches. In the first approach, the cost of performing a CGDA execution step was reduced with the introduction of state of the art techniques that reduce the number of evaluations. In the second approach, a new evolutionary strategy has been proposed to increase the update frequency of the environment within a CGDA execution step. A concrete implementation of this strategy, OET, was able to achieve replanning times of few seconds, similar to the ones observed in humans.
- **Style Transfer with a Deep Reinforcement Learning controller for its application in robotic actions**. Two different frameworks have been proposed as a way to introduce Neural Style Transfer and DRL in robotic actions. First, the NPST framework is designed to work with environments involving discrete action spaces and without requiring the introduction of an additional loss network. Then, the NPST3 framework was proposed as a more advanced version of NPST with the introduction of continuous action spaces and an Autoencoder network to define the Content and the Style. The NPST3 is additionally designed to allow the online teleoperation of the robot with the online introduction of the Style within the teleoperated trajectory.

8.3 Future Work

Two different areas of study are introduced for presenting the future works related to the contributions proposed in this thesis. The first area of study focuses on studying the improvement of the proposed methods introduced in this thesis. In the second area, future applications that take advantage of the methods proposed in this thesis are presented.

8.3.1 Proposed Enhancements

Different improvements can be proposed as a way to enhance the performance of the proposed methods, increase the generalization capabilities of the proposed framework, or lead to a deeper study that can be useful for future applications.

- **Automatic constraint definition**. The introduction of a method to automatically extract constraints to introduce in the CGDA framework using human demonstrations.
- **Further reductions in RIT for the OET algorithm**. This includes the introduction of parallel computation.
- **Increasing the generalization capabilities of NPST3**. One network has to be trained for each selected Style. Future works include the study of a loss network that is able to work with multiple Styles.
- **Experiments with social robotics platforms**. Experimental results using the NPST3 framework include a manipulator for the introduction of the selected human emotions defined as the Styles for these experiments. Further studies should introduce a social robot platform as an, initially, more suitable platform for the introduction of this kind of Styles.

- **Increase the range of studied Styles**. Styles in robotics do not need to be limited to human emotions. Further works include the definition of new Styles as, for example, the Styles defined by an expert human operator. This would allow an inexperienced operator to command the robot with the Style of an expert.

8.3.2 Proposed Applications

Due to the theoretical nature of this thesis, future works also include a wide range of possible applications derived from the frameworks proposed. Some of these possible future applications are proposed in this section. These applications introduce the advantages proposed by the different contributions presented in this thesis.

- **The introduction of CGDA within Smart city applications.** This idea was already presented in Fernandez-Fernandez et al. (85). Thanks to the improvements proposed in this thesis, the introduction of Learning from Demonstration and Action Generalization makes CGDA suitable for applications involving non-technical users in dynamic environments. This is the case, for example, of Smart City applications.
- **Collaborative tasks with human subjects.** The introduction of fast replanning times using the OET algorithm allows the proposition of new applications using CGDA, e.g. introducing human subjects to perform collaborative action executions.
- **Social robot applications**. The automatic introduction of emotions in robots allows the introduction of new applications in social robotics. The NPST3 framework allows the operator to command a social robot while the framework takes care of introducing the low-level features to make the robot move while expressing the selected human emotion.

- **Shared-control Style Transfer applications**. NPST3 allows the online transferring of Styles to the commanded motions defined by the user in a similar way to classical shared control applications. Future applications introducing shared control architectures and Style Transfer can be proposed based on this idea.

References

References

1. Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423. IEEE, 2016. doi: 10.1109/CVPR.2016.265.

2. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. doi: 10.1038/nature14236.

3. Santiago Morante, Juan G Victores, Alberto Jardón, and Carlos Balaguer. Humanoid robot imitation through continuous goal-directed actions: an evolutionary approach. *Advanced Robotics*, 29(5):303–314, 2015. doi: 10.1080/01691864.2014.964314.

4. David Hume. *A treatise of human nature*. Courier Corporation, 2003.

5. IEP Staff. Deductive and inductive arguments. the internet encyclopedia of philosophy, 2016. ISSN 2161-0002.

6. D Herms. Logical basis of hypothesis testing in scientific research. In *A logic primer to accompany Giere*. Ohio State University, 1984.

7. Janet L Kolodner. An introduction to case-based reasoning. *Artificial intelligence review*, 6(1):3–34, 1992. doi: 10.1007/BF00155578.

8. Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009. ISSN 0921-8890. doi: 10.1016/j.robot.2008.10.024.

9. Sylvain Calinon, Florent Guenter, and Aude Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2): 286–298, 2007. doi: 10.1109/TSMCB.2006.886952.

10. Karinne Ramirez-Amaro, Michael Beetz, and Gordon Cheng. Transferring skills to humanoid robots by extracting semantic representations from observations of human activities. *Artificial Intelligence*, 247:95–118, 2017. ISSN 0004-3702. doi: 10.1016/j.artint.2015.08.009.

11. Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, Lorenz Mösenlechner, Dejan Pangercic, Thomas Rühr, and Moritz Tenorth. Robotic roommates making pancakes. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 529–536. IEEE, 2011. doi: 10.1109/Humanoids.2011.6100855.

12. Daniel Holden, Ikhsanul Habibie, Ikuo Kusajima, and Taku Komura. Fast neural style transfer for motion data. *IEEE Computer Graphics and Applications*, 37(4):42–49, 2017. ISSN 02721716. doi: 10.1109/MCG.2017.3271464.

13. Juan G Victores, Santiago Morante, Alberto Jardón, and Carlos Balaguer. Towards robot imagination through object feature inference. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5694–5699. IEEE, 2013. doi: 10.1109/IROS.2013.6697181.

14. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR*, 2015.

15. A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Survey: Robot programming by demonstration. *Springer Handbook of Robotics*, pages 1371–1394, 2008. doi: 10.1007/978-3-540-30301-5_60.

16. Baris Akgun, Maya Cakmak, Jae Wook Yoo, and Andrea Lockerd Thomaz. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 391–398. IEEE, 2012. doi: 10.1145/2157689.2157815.

17. John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992. ISBN 9780262581110. doi: 10.7551/mitpress/1090.001.0001.

18. Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. ISBN 0262039249. doi: 10.5555/3312046.

19. Sylvain Calinon and Aude Billard. Recognition and reproduction of gestures using a probabilistic framework combining PCA, ICA and HMM. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 105–112. Association for Computing Machinery, 2005. ISBN 1595931805. doi: 10.1145/1102351.1102365.

20. Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013. doi: 10.1162/NECO_a_00393.

21. Chrystopher L Nehaniv and Kerstin Dautenhahn. Of hummingbirds and helicopters: An algebraic framework for interdisciplinary studies of imitation and its applications. In *Interdisciplinary approaches to robot learning*, volume 24, pages 136–161. World Scientific, 2000. ISBN 978-981-279-274-7. doi: 10.1142/9789812792747_0007.

22. Santiago Morante, Juan G Victores, and Carlos Balaguer. Automatic demonstration and feature selection for robot learning. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 428–433. IEEE, 2015. doi: 10.1109/HUMANOIDS.2015.7363569.

23. Meinard Müller. *Dynamic Time Warping*, pages 69–84. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-74048-3. doi: 10.1007/978-3-540-74048-3_4.

24. Martin D Buhmann. Radial basis functions. *Acta numerica*, 9:1–38, 2000. doi: 10.1017/CBO9780511543241.

25. Yaochu Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft computing*, 9(1):3–12, 2005. doi: 10.1007/s00500-003-0328-5.

26. Loris Vincenzi and Marco Savoia. Improving the speed performance of an evolutionary algorithm by a second-order cost function approximation. In *Proceedings of the 2nd International Conference on Engineering Optimization*, pages 6–9. APMTAC, 2010. ISBN 978-989-96264-3-0.

27. Robert E. Smith, B. A. Dike, and S. A. Stegmann. Fitness inheritance in genetic algorithms. In *Proceedings of the 1995 ACM Symposium on Applied Computing*, SAC '95, page 345–350. Association for Computing Machinery, 1995. ISBN 0897916581. doi: 10.1145/315891.316014.

28. Robert Barbour, David Corne, and John McCall. Accelerated optimisation of chemotherapy dose schedules using fitness inheritance. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010. doi: 10.1109/CEC.2010.5586118.

29. Anna I Esparcia-Alcázar and Jaroslav Moravec. Fitness approximation for bot evolution in genetic programming. *Soft Computing*, 17(8): 1479–1487, 2013. doi: 10.1007/s00500-012-0965-7.

30. Ilaria Bertini, Matteo De Felice, Alessandro Pannicelli, and Stefano Pizzuti. Soft computing based optimization of combined cycled power plant start-up operation with fitness approximation methods. *Applied Soft Computing*, 11(6):4110–4116, 2011. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2011.02.028.

31. James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995. doi: 10.1109/ICNN.1995.488968.

32. Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007. doi: 10.1007/s11721-007-0002-0.

33. Mohsen Davarynejad, M-R Akbarzadeh-T, and Naser Pariz. A novel general framework for evolutionary optimization: Adaptive fuzzy fitness granulation. In *2007 IEEE Congress on Evolutionary Computation*, pages 951–956. IEEE, 2007. doi: 10.1109/CEC.2007.4424572.

34. M-R Akbarzadeh-T, Mohsen Davarynejad, and Naser Pariz. Adaptive fuzzy fitness granulation for evolutionary optimization. *International Journal of Approximate Reasoning*, 49(3):523–538, 2008. doi: 10.1016/j.ijar.2008.05.004.

35. Els Ducheyne, Bernard De Baets, and Robert De Wulf. Is fitness inheritance useful for real-world applications? In *International Conference on Evolutionary Multi-Criterion Optimization*, EMO'03, pages 31–42. Springer, 2003. ISBN 3540018697.

36. Margarita Reyes-Sierra and Carlos A Coello Coello. Fitness inheritance in multi-objective particle swarm optimization. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pages 116–123. IEEE, 2005. doi: 10.1109/SIS.2005.1501610.

37. Margarita Reyes-Sierra and Carlos A Coello Coello. A study of fitness inheritance and approximation techniques for multi-objective particle swarm optimization. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 65–72. IEEE, 2005. doi: 10.1109/CEC.2005.1554668.

38. Kusum Deep and Dipti. A self-organizing migrating genetic algorithm for constrained optimization. *Applied Mathematics and Computation*, 198(1):237–250, 2008. ISSN 0096-3003. doi: https://doi.org/10.1016/j.amc.2007.08.032.

39. Yong Zhao and Nobuo Sannomiya. An improvement of genetic algorithms by search space reductions in solving large-scale flowshop problems. *IEEJ Transactions on Electronics, Information and Systems*, 121(6):1010–1015, 2001. doi: 10.1541/ieejeiss1987.121.6_1010.

40. Piya Chootinan and Anthony Chen. Constraint handling in genetic algorithms using a gradient-based repair method. *Computers & operations research*, 33(8):2263–2281, 2006. ISSN 0305-0548. doi: 10.1016/j.cor.2005.02.002.

41. Özgür Yeniay. Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and computational Applications*, 10(1):45–56, 2005. ISSN 2297-8747. doi: 10.3390/mca10010045.

42. Atidel Ben Hadj-Alouane and James C Bean. A genetic algorithm for the multiple-choice integer program. *Operations research*, 45(1):92–101, 1997. doi: 10.1287/opre.45.1.92.

43. Zbigniew Michalewicz and Naguib Attia. Evolutionary optimization of constrained problems. In *Proceedings of the 3rd annual conference on evolutionary programming*, pages 98–108. World Scientific, 1994.

44. Jianhua Xiao, Jin Xu, Zehui Shao, Congfeng Jiang, and Linqiang Pan. A genetic algorithm for solving multi-constrained function optimization problems based on KS function. In *2007 IEEE Congress on Evolutionary Computation*, pages 4497–4501. IEEE, 2007. doi: 10.1109/CEC.2007.4425060.

45. Raziyeh Farmani and Jonathan A Wright. Self-adaptive fitness formulation for constrained optimization. *IEEE transactions on evolutionary computation*, 7(5):445–455, 2003. doi: 10.1109/TEVC.2003.817236.

46. Simone Puzzi and Alberto Carpinteri. A double-multiplicative dynamic penalty approach for constrained evolutionary optimization. *Structural and Multidisciplinary Optimization*, 35(5):431–445, 2008. doi: 10.1007/s00158-007-0143-1.

47. Biruk Tessema and Gary G Yen. An adaptive penalty formulation for constrained evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 39(3):565–578, 2009. doi: 10.1109/TSMCA.2009.2013333.

48. Li-Chiu Chang. Guiding rational reservoir flood operation using penalty-type genetic algorithm. *Journal of Hydrology*, 354(1):65–74, 2008. ISSN 0022-1694. doi: 10.1016/j.jhydrol.2008.02.021.

49. Gilbert Syswerda. A study of reproduction in generational and steady-state genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pages 94–101. Elsevier, 1991. doi: 10.1016/B978-0-08-050684-5.50009-4.

50. Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, 2010.

51. Paul Fitzpatrick, Giorgio Metta, and Lorenzo Natale. Towards long-lived robot genes. *Robotics and Autonomous systems*, 56(1):29–45, 2008. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2007.09.014.

52. Santiago Martínez, Concepción Alicia Monje, Alberto Jardón, Paolo Pierro, Carlos Balaguer, and Delia Munoz. Teo: Full-size humanoid robot design powered by a fuel cell system. *Cybernetics and Systems*, 43(3):163–180, 2012. doi: 10.1080/01969722.2012.659977.

53. John G. Avildsen, Bill Conti, and Brooks Arthur. The karate kid, 1984.

54. Stjepan Picek, Marin Golub, and Domagoj Jakobovic. Evaluation of crossover operator performance in genetic algorithms with binary representation. In *Proceedings of the 7th International Conference on Intelligent Computing: Bio-Inspired Computing and Applications*, ICIC'11, pages 223–230. Springer, 2011. ISBN 9783642245527. doi: 10.1007/978-3-642-24553-4_31.

55. Jessica B Hamrick, Kevin A Smith, Thomas L Griffiths, and Edward Vul. Think again? the amount of mental simulation tracks uncertainty in the outcome. In *Proceedings of the 37th Annual Meeting of the Cognitive Science Society*, 2015. doi: 10.6084/M9.FIGSHARE.1554893.V1.

56. Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. ISSN 1532-4435. doi: 10.5555/2946645.2946684.

57. Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017. doi: 10.1109/ICRA.2017.7989385.

58. Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. In *Robotics: Science and Systems*, 2019. doi: 10.15607/RSS.2019.XV.011.

59. 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. In *International Conference on Learning Representations (ICLR)*, 2016.

60. Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 1587–1596. PMLR, 2018.

61. Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30 of *AAAI'16*, page 2094–2100. AAAI Press, 2016.

62. Joshua B Tenenbaum and William T Freeman. Separating style and content. *Advances in neural information processing systems*, 9, 1997.

63. Armin Bruderlin and Lance Williams. Motion signal processing. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 97–104, 1995. ISBN 0897917014. doi: 10.1145/218380.218421.

64. Munetoshi Unuma, Ken Anjyo, and Ryozo Takeuchi. Fourier principles for emotion-based human figure animation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 91–96. Association for Computing Machinery, 1995. ISBN 0897917014. doi: 10.1145/218380.218419.

65. Kenji Amaya, Armin Bruderlin, and Tom Calvert. Emotion from motion. In *Graphics interface*, volume 96, pages 222–229. Toronto, Canada, 1996. ISBN 9781315579214. doi: 10.4324/9781315579214.

66. Jianyuan Min, Huajun Liu, and Jinxiang Chai. Synthesis and editing of personalized stylistic human motion. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, I3D '10, pages 39–46. Association for Computing Machinery, 2010. doi: 10.1145/1730804.1730811.

67. Megha Sharma, Dale Hildebrandt, Gem Newman, James E Young, and Rasit Eskicioglu. Communicating affect via flight path exploring use of the laban effort system for designing affective locomotion paths. In *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 293–300. IEEE, 2013. doi: 10.1109/HRI.2013.6483602.

68. Allan Zhou and Anca D Dragan. Cost functions for robot motion style. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3632–3639. IEEE, 2018. doi: 10.1109/IROS.2018.8594433.

69. Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2242–2251. IEEE, 2017. doi: 10.1109/ICCV.2017.244.

70. Tianying Wang, Wei Qi Toh, Hao Zhang, Xiuchao Sui, Shaohua Li, Yong Liu, and Wei Jing. Robocodraw: Robotic avatar drawing with gan-based style transfer and time-efficient path optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:10402–10409, 2020. doi: 10.1609/aaai.v34i06.6609.

71. Yaxin Liu, Xiaoyan Zhang, and Xiaogang Xu. Semantic-aware video style transfer based on temporal consistent sparse patch constraint. In *2021 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2021. doi: 10.1109/ICME51207.2021.9428352.

72. Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009. doi: 10.1109/CVPR.2009.5206848.

73. Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. *International Conference on Learning Representations (ICLR)*, 2017.

74. Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep painterly harmonization. *Computer Graphics Forum*, 37(4):95–106, 2018. ISSN 14678659. doi: 10.1111/cgf.13478.

75. Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. Style transfer in text: Exploration and evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32. AAAI Press, 2018.

76. Joseph Lee, Ziang Xie, Cindy Wang, Max Drach, Dan Jurafsky, and Andrew Y Ng. Neural text style transfer via denoising and reranking. In *Proceedings of the Workshop on Methods for Optimizing and Evaluating Neural Language Generation*, pages 74–81. Association for Computational Linguistics, 2019. doi: 10.18653/v1/W19-2309.

77. Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, number 346 in ICML ’04, page 1. Association for Computing Machinery, 2004. ISBN 1581138285. doi: 10.1145/1015330.1015430.

78. Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint*, 2015.

79. Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd national conference on Artificial intelligence-Volume 3*, AAAI’08, pages 1433–1438. AAAI Press, 2008. ISBN 9781577353683. doi: 10.5555/1620270.1620297.

80. Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, 1997. ISSN 0098-3500. doi: 10.1145/279232.279236.

81. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

82. Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814. Omnipress, 2010. ISBN 9781605589077. doi: 10.5555/3104322.3104425.

83. Yanran Li, Zhao Wang, Xiaosong Yang, Meili Wang, Sebastian Iulian Poiana, Ehtzaz Chaudhry, and Jianjun Zhang. Efficient convolutional hierarchical autoencoder for human motion prediction. *The Visual Computer*, 35(6):1143–1156, 2019. doi: 10.1007/s00371-019-01692-9.

84. Éric Marchand, Fabien Spindler, and François Chaumette. ViSP for visual servoing: a generic software platform with a wide class of robot control skills. *IEEE Robotics & Automation Magazine*, 12(4):40–52, 2005. doi: 10.1109/MRA.2005.1577023.

85. Raul Fernandez-Fernandez, Juan G Victores, David Estevez, and Carlos Balaguer. Real evaluations tractability using continuous goal-directed actions in smart city applications. *Sensors*, 18(11), 2018. ISSN 1424-8220. doi: 10.3390/s18113818.