Universidad Carlos III de Madrid

DOCTORAL THESIS

CONTINUOUS GOAL-DIRECTED ACTIONS: ADVANCES IN ROBOT LEARNING

Santiago Morante Cendrero

Advisor: Dr. Juan Carlos González Víctores

Advisor: Dr. Carlos Balaguer Bernaldo de Quirós

Ph.D program in Electrical Engineering, Electronics and Automation

Robotics Lab Research Group

Dept. Systems Engineering and Automation

Universidad Carlos III de Madrid

Leganés, December 2015

UNIVERSIDAD CARLOS III DE MADRID

Ph.D program in Electrical Engineering, Electronics and Automation

The review committee approves the doctoral thesis titled "**CONTINUOUS GOAL-DIRECTED ACTIONS: ADVANCES IN ROBOT LEARNING**" written by **Santiago Morante Cendrero** and directed by **Dr. Juan Carlos González Víctores** and **Dr. Carlos Balaguer Bernaldo de Quirós**.

Signature of the Review Committee:

President: _____

Secretary: _____

Member: _____

Grade:

Date:

"It took time, but I learned the patterns. At first, the sheer complexity made the task seem impossible. But then I felt the satisfaction of breaking the code"

Daniel H. Wilson, Robogenesis

Acknowledgments

First, I would like to thank Juan Carlos Gonzalez Victores. He has been more than an advisor. He has been my friend during the journey, helped me with the "software stuff", amazed me with his programming skills, and wrote (and reviewed) with me every paper we published. Thank you Johnny!

I also would like to express my gratitude to my advisor Carlos Balaguer for supporting me and for letting me deviate from my research topic more often than not. I can't thank enough to my Robotics Lab colleagues, specially Álvaro Villoslada, Roberto Montero, and Jorge Muñoz, for being so easy going and kind.

Finally, I will like to thank my parents for their support, and Lourdes, because she is my pillar of strength.

Abstract

Robot Programming by Demonstration (PbD) has several limitations. This thesis proposes a solution to the shortcomings of PbD with an inspiration on Goal-Directed imitation applied to robots. A framework for goal imitation, called Continuous Goal-Directed Actions (CGDA), has been designed and developed. This framework provides a mechanism to encode actions as changes in the environment. CGDA learns the objective of the action, beyond the movements made to perform it. With CGDA, an action such as "painting a wall" can be learned as "the wall changed its color a 50% from blue to red". Traditional robot imitation paradigms such as PbD would learn the same action as "move joint *i 30* degrees, then joint *j* 43 degrees...".

This thesis' main contribution is innovative in providing a framework able to measure and generalize the effects of actions. It also innovates by creating metrics to compare and reproduce goal-directed actions. Reproducing actions encoded in terms of goals allows a robot-configuration independence when reproducing tasks. This innovation allows to circumvent the correspondence problem (adapting the kinematic parameters from humans to robots).

CGDA can complement current kinematic-focused paradigms, such as PbD, in robot imitation. CGDA action encoding is centered on the changes an action produces on the features of objects altered during the action. The features can be any measurable characteristic of the objects such as color, area, form, etc. By tracking object features during human action demonstrations, a high dimensional feature trajectory is created. This trajectory represents a finely-grained sequence of object temporal states during the action. This trajectory is the main resource for the generalization, recognition and execution of actions in CGDA.

Around this presented framework, several components have been added to facilitate and improve the imitation. Naïve implementations of robot learning frameworks usually assume that all the data from the user demonstrations has been correctly sensed and is relevant to the task. This assumption proves wrong in most human-demonstrated learning scenarios. This thesis presents an automatic demonstration and feature selection process to solve this issue. This machine learning pipeline is called Dissimilarity Mapping Filtering (DMF). DMF can filter both irrelevant demonstrations and irrelevant features.

Once an action is generalized from a series of correct human demonstrations, the robot must be provided a method to reproduce this action. Robot joint trajectories are computed in simulation using evolutionary computation through diverse proposed strategies. This computation can be improved by using human-robot interaction. Specifically, a system for robot discovery of motor primitives from random human-guided movements has been developed. These Guided Motor Primitives (GMP) are combined to reproduce goal-directed actions.

To test all these developments, experiments have been performed using a humanoid robot in a simulated environment, and the real full-sized humanoid robot TEO. A brief analysis about the cyber safety of current robots is additionally presented in the final appendices of this thesis.

Keywords: robot learning, humanoid robots, goal-directed actions, motor primitives, feature selection, demonstration selection, cryptobotics.

Resumen

Robot Programming by demonstration (PbD) tiene varias limitaciones. Esta tesis propone una solución a las carencias de PbD, inspirándose en la imitación dirigida a objetivos en robots. Se ha diseñado y desarrollado un marco de trabajo para la imitación de objetivos llamado Continuous Goal-Directed Actions (CGDA). Este marco de trabajo proporciona un mecanismo para codificar acciones como cambios en el entorno. CGDA aprende los objetivos de la acción, mas allá de los movimientos hechos para realizarla. Con CGDA, una acción como "pintar una pared"se puede aprender como "la pared cambió su color un 50 % de azul a rojo". Paradigmas tradicionales de imitación robótica como PbD aprenderían la misma acción como "mueve la articulación i 30 grados, luego la articulación j 43 grados...".

La contribución principal de esta tesis es innovadora en proporcionar un marco de trabajo capaz de medir y generalizar los efectos de las acciones. También innova al crear metricas para comparar y reproducir acciones dirigidas a objetivos. Reproducir acciones codificadas como objetvos permite independizarse de la configuracion del robot cuando se reproducen las acciones. Esta innovación permite sortear el problema de la correspondencia (adaptar los parámetros cinemáticos de los humanos a los robots).

CGDA puede complementar paradigmas centrados en la cinemática, como PbD, en la imitación robótica. CGDA codifica las acciones centrándose en los cambios producidos por la acción en las características de los objetos afectados por ésta. Las características pueden ser cualquier rasgo medible de los objetos, como color, area, forma, etc. Midiendo las características de los objetos durante las demostraciones humanas se crea una trayectoria de alta dimensionalidad. Esta trayectoria representa una detallada secuencia de los estados temporales del objeto durante la acción. Esta trayectoria es el

recurso principal para la generalización, el reconocimiento y la ejecución de acciones en CGDA.

Alrededor del marco de trabajo presentado, se han añadido algunos componentes para facilitar y mejorar la imitación. Las implementaciones simples en aprendizaje robótico normalmente asumen que todos los datos provenientes de las demostraciones del usuario han sido correctamente medidos y son relevantes para la tarea. Esta suposición se demuestra falsa en la mayoría de escenarios de aprendizaje por demostración humana. Esta tesis presenta un proceso de selección automático de demostraciones y caracteristicas para resolver este problema. Este proceso de aprendizaje automático se llama Dissimilarity Mapping Filtering (DMF). DMF puede filtrar tanto demostraciones irrelevantes, como características innecesarias.

Una vez que una acción se ha generalizado a partir de una serie de demostraciones humanas, es necesario proveer al robot de un método para reproducir la acción. Las trayectorias articulares del robot se computan en simulación usando computación evolutiva. Esta computación se puede mejorar usando interacción humano-robot. Especificamente, se ha desarrollado un sistema para el descubrimiento de primitivas de movimiento del robot a partir de movimientos aleatorios, guiados por el humano. Estas primitivas, llamadas Guided Motor Primitives (GMP), se conbinan para reproducir acciones centradas en objetivos.

Para probar estos desarrollos, los experimentos se han llevado a cabo usando un robot humanoide en un entorno simulado, y el robot humanoide real TEO. En los apéndices finales de esta tesis se presenta un breve análisis de la ciberseguridad de los robots actuales .

Palabras clave: aprendizaje robótico, robots humanoides, acciones centradas en objetivos, primitivas de movimiento, selección de características, selección de demostraciones, cryptobotics.

Contents

Acknowledgments **vii**

Abstract **ix**

Resumen **xi**

Contents **xiii**

Index of Figures **xvii**

Index of Tables **xxiii**

1 Introduction **1**

1.1 New Mechanisms for Programming Robots 1
1.2 Overview and Definitions 2
1.3 Motivation 4
1.4 Main Objectives 6
1.5 Scientific Contributions 6
1.6 Document Structure 7

2 Background **9**

2.1 Programming by Demonstration 9
2.2 Goal-Directed Actions 11
2.3 Demonstration and Feature Selection 15
2.4 Motor Primitives 18

2.5 Physical Interaction with Robots 20
2.6 Chapter Summary 23

3 Continuous Goal-Directed Actions **25**
3.1 Differences between CGDA and PbD 26
3.2 Selecting Relevant Demonstrations and Features 29
3.3 Basic CGDA Framework 29
3.4 Generating a Library of Motor Primitives 30
3.5 Compensating Friction and Gravity 31

4 Demonstration and Feature Selection **33**
4.1 Dissimilarity Mapping Filtering 35
4.2 Demonstration Selector 35
4.2.1 Preprocessing 37
4.2.2 Dissimilarity 38
4.2.3 Mapping 41
4.2.4 Filtering 41
4.3 Feature Selector 42
4.3.1 Preprocessing 43
4.3.2 Dissimilarity 43
4.3.3 Mapping 44
4.3.4 Filtering 45

5 Generalization and Recognition **47**
5.1 Generalization 47
5.1.1 Time Rescaling 48
5.1.2 Average in Temporal Intervals 48
5.1.3 Radial Basis Function Interpolation 48
5.2 Recognition 51
5.2.1 Dynamic Time Warping 51
5.2.2 Euclidean Distance 52

6 Execution **55**
6.1 Simulation using Evolutionary Strategies 56
6.1.1 Full Trajectory Evolution 57

6.1.2 Individual Evolution 57
6.1.3 Incrementally Evolved Trajectories 57
6.2 Guided Motor Primitives 58
6.2.1 Sequential Incremental Combinatorial Search 61
6.2.2 Force-Sensorless Friction and Gravity Compensation 65

7 Experiments **73**
7.1 Demonstration and Feature Selection 74
7.1.1 Demonstration Selector 75
7.1.2 Feature Selector 78
7.2 CGDA Recognition 80
7.2.1 Object Feature Trajectory Recognition 80
7.2.2 Cartesian Space Trajectory Recognition 82
7.3 CGDA Execution by Evolutionary Computation 82
7.3.1 Executing a Spatial Task: Cleaning 84
7.3.2 Executing a Non-Spatial Task: Painting 86
7.4 Guided Motor Primitives 91
7.4.1 Executing a Spatial Task: Cleaning 91
7.4.2 Executing a Non-Spatial Task: Painting 92
7.5 Friction and Gravity Compensation 94
7.5.1 Computation 95
7.5.2 Evaluation 97

8 Discussion and Conclussions **99**
8.1 Normalization on Demonstration and Feature Selection 99
8.2 Limitations of CGDA 100
8.3 The Benefits of Human-Robot Interaction 101
8.4 A More Accurate Compensation 101
8.5 Conclusions 102
8.6 Future Lines of Research 103

A Opinion: Cryptobotics **107**
A.1 Stating The Problem 107
A.2 Current State of Security in Mainstream Robotic Software 110

A.3 Guidelines and Security Checks 112
A.4 Discussion 114

B TEO, The Humanoid Robot **115**

Bibliography **125**

Index of Figures

1.1 Continuous Goal-Directed Actions sketch: From a set task demonstrations (also called repetitions), a goal model is created. Then, action reproduction (also called execution) can be performed. 3

1.2 This figure represents the beginning and the end of an imitation scenario. It starts with a series of task repetitions perform by a human and recorded by the robot. It ends with the robot reproducing the task by itself. 5

2.1 The framework usually starts with a human performing several demonstration of the task. From this point, a generalization process generates a task model, in human joint terms. Then, the correspondence problem is solved, to some extent, to translate the task into the robot morphology. Finally, the task can be reproduced in robot joint terms. 10

2.2 PbD starts with a human performing a task. Kinematic parameters are recorded for a later generalization. 10

2.3 The goal-directed framework usually starts with a human performing several demonstration of the task. From this point, a generalization process generates a goal model, in terms of the consequences of the action in the environment. Then, the execution problem is solved to translate the goal model into a kinematic model. Finally, the task can be reproduced in robot joint terms. 12

2.4 Goal imitation starts with a human performing a task. Object features are recorded for a later generalization. In the goal-only case, only the start and the end state are recorded. In CGDA case, the full sequence of states are recorded. 13
2.5 In the upper block, demonstration selection is carried out. From the set of complete demonstrations (each one represented as an axis frame with blue lines), the most different are discarded. In the lower block, feature selection is carried out. From the set of features, those feature which are not similar across demonstration are removed. 16
2.6 From a set of primitives, a combination which accomplishes the goal is extracted. 19
2.7 Without compensation, robots are very stiff to be moved. With compensation, even a partial one, robots can be guided easily. 21

3.1 Block diagram of the thesis. The block order defines the temporal order of the process. First, a demonstration and feature selection is performed. Then comes the generalization. The next step is to be able to perform recognition, or comparison, among feature trajectories. Finally, and using the information from the previous blocks, action reproduction can be completed. There are two main alternatives for the joint parameters generation: to use evolutionary computation or to combine guided motor primitives. This last option uses the friction and gravity compensation controllers developed. 27

4.1 Demonstration and Feature Selection block. 33
4.2 Imagine we want to screen the most irrelevant of the three signals. First, we extract a measure of dissimilarities among them. These dissimilarities can be represented as nodes' distances in a graph. Then, we map these distances from a matrix to a single dimension e.g. by summing the columns. In the last step, we can represent values in a bar plot. To further increase their differences, a transformation, e.g. z-score, is applied. This transformation allows to better distinguish the irrelevant signal, which can now be filtered with a threshold (represented by a dotted red line). . 36
4.3 Humanoid robotics learning scenario: block-moving task. 36

4.4 Different preprocessing methods for reducing multidimensional information. 39
4.5 Example of accumulated cost matrix for two sequences. On the left side, the interpolated sequences are shown. The right side depicts the Dynamic Time Warping computation. White cells represent high cost, while dark cells are low cost ones. The red line is the lowest cost path. 40
4.6 For each feature space, a distance matrix is calculated. Each color line (red, blue and green) is the same feature but extracted from a different demonstration. When represented in the same space, these features trajectories will show some kind of similarity if they are relevant, or will be different if they are not. 44

5.1 Generalization block. 47
5.2 Plot representing a three feature trajectory. Black lines are training action repetitions. The blue dots are the averaged points. 49
5.3 Plot representing a three feature trajectory. Black lines are training action repetitions. The blue line is the generalization of all the repetitions. . . . 50
5.4 Recognition block. 51
5.5 Two alternatives are presented to be used as distance measures. 53

6.1 Execution block. 55
6.2 Evolutionary block. 56
6.3 Guided Motor Primitives block. 59
6.4 Human guiding the robot in a random spatial trajectory. The red lines represent an example of the joints' spatial trajectories during the guiding. 59
6.5 This random joint trajectory involves 3 joints of the robot arm. It is generated when the human performs some random movements with the robot arm. 60
6.6 An example of comparison between two random segments extracted from the previously shown joint trajectory. Both segments have a duration of $\tau = 0.5$ s and the blue lines represent the distances between pairs of points. 61
6.7 Several examples of guided motor primitives in the joint space. They were extracted from the joint trajectory shown in Figure 6.5. 62

6.8 SICS example for a two-point feature trajectory. For the first point, the nodes are expanded and a breadth-first search is performed until the cost of one of the nodes is lower than a threshold ε. If this occurs, the path to the node is stored and the process continues for the next point. If not, the tree is expanded one level of depth, and the search repeats. 64
6.9 Friction and Gravity Compensation block. 65
6.10 Block scheme of friction and gravity compensation. In this type of control, there is no external reference, the robotic system is moved by the external perturbations caused by the user. 66
6.11 This friction model includes Coulomb friction and viscous friction. The variables α and β represent the parameters of the linear model assumed. 69

7.1 User demonstrations from TEO's perspective; red segmentation on the bottom left, and green segmentation on the bottom right. 74
7.2 Red object centroid coordinates throughout time. 76
7.3 Green object centroid coordinates throughout time. 77
7.4 Demonstration selection's z-scores without data normalization, preprocessed by summing columns, setting the threshold to $\alpha = 0.5$. 77
7.5 Demonstration selection's z-scores without data normalization, preprocessed by summing rows, setting the threshold to $\alpha = 0.5$. 78
7.6 Feature selection's z-scores without data normalization, setting the threshold to $\alpha = 0.5$. 79
7.7 Graphical representation of the actions performed in this experiment. . . 80
7.8 The picture shows the experimental scenario with the robot, the object (green) and the Kinect camera. The bottom left square is the Kinect depth map and the bottom right square shows the color segmentation of the object. 83
7.9 Fitness value through evolution. The red point is the minimum value achieved by evolution. 85
7.10 Unidimensional temporal plots of generalized reference (blue), and the object feature space trajectory from executing the EC winner joint position trajectory (red). The Z dimension gives the worst results. The system was not able to reduce the error in this dimension. 86

7.11 Unidimensional temporal plots of the generalized action, and the trajectory obtained by executing the EC winner joint parameters. The trajectories are quite similar for all cases. In the case of Z dimension, which may look like a less well-defined result, it is important to check its units. In none of the points the error is bigger than 20 mm. 87

7.12 Plot representing a three feature trajectory. Black lines are training action repetitions. The blue line is the generalization extracted from all the repetitions. The red line is the evolved trajectory. Note that none of the points of the evolved trajectory has an error bigger than 20 mm with respect to its reference (the generalized). 88

7.13 Action execution for a CLEAN command. Despite the trajectory is purely spatial, the objective of the evolution and the execution is to perform a feature trajectory obtained from the object tracker. 88

7.14 In this chart the time taken to compute each point is shown. Except for one final point (where the valid space is already very restricted) the time of computation increases linearly. 89

7.15 Unidimensional temporal plot of the generalized action, and the trajectory obtained by executing the EC winner joint parameters. As seen for a linear painting task (one wall square is painted in each step), the performance is very accurate. 90

7.16 Action execution for a PAINT command. Each square changes its color when the hand gets closer than a specific distance. 90

7.17 Cleaning: Plot superposing the generalized feature trajectory (blue) and several feature trajectories resulting from executing the sequence of primitives selected by SICS. Results are dependent on the number of primitives available to choose from (given by the threshold of similarity ξ) and the maximum tree depth d. The primitives used belong to the same set of segments, but in function of ξ, the most diverse are selected. 92

7.18 Painting: Plot superposing the generalized feature trajectory (blue) and several feature trajectories resulting from executing the sequence of primitives selected by SICS. Results are dependent on the number of primitives available to choose from (given by the threshold ξ). Remember the plot is not representing joint or spatial trajectories, but feature ones. . . . 93

7.19 Several screenshots taken during the painting experiment. Motor sequence obtained from combining primitives using SICS. 94
7.20 Velocity vs. time curves, with constant torques applied in each curve. There is a proportional linear dependence between current in the motor and torque applied. 96
7.21 Stabilization velocity vs. the torque applied. The Coulomb friction (blue) and the viscous friction (red) can be seen. The same procedure is applied to negative velocities. 96
7.22 Velocity profiles for several initial 'pushes' with the Zero-Friction Zero-Gravity controller (ZFZG). 98
7.23 Sequence of the movement of the robot arm using a Zero-Friction Zero-Gravity controller. When the user pushes the arm, it moves freely in the direction of the applied force. 98

B.1 Axes and directions of rotation of humanoid robot TEO. 116
B.2 Denavit-Hartenberg directions of humanoid robot TEO. 117
B.3 Denavit-Hartenberg parameters of humanoid robot TEO. 118
B.4 CAN (Controller Area Network) identification number of the joints of humanoid robot TEO. 119
B.5 YARP identification number of the joints of humanoid robot TEO. 120
B.6 Mechanical links lengths distribution of humanoid robot TEO. 121
B.7 Mechanical links lengths measurements of humanoid robot TEO. 122
B.8 Position of the Center of Masses in the concentrated mass model of humanoid robot TEO. 123

Index of Tables

3.1 Main differences between PbD and CGDA paradigms. 28

7.1 z-scores with normalized data. A green cell means the feature is accepted. A red cell means the feature is discarded. Threshold is set to $\alpha = 0.5$. 79

7.2 DTW cost matrix: test actions (lower case) vs. generalized trajectories (upper case). Bold numbers represent best values (minimum discrepancy). 81

7.3 DTW cost matrix: reduced test actions (lower case) vs. reduced generalized trajectories (upper case). Bold numbers represent best values (minimum discrepancy). Only spatial features are used. 81

7.4 DTW cost matrix: Cartesian test actions (lower case) vs. Cartesian generalized trajectories (upper case). Bold numbers represent best values (minimum discrepancy). 82

Chapter 1

Introduction

This chapter introduces the evolution in robot programming through time. It also provides definitions for common terms used in the thesis. Motivation and objectives are outlined. Scientific contributions in the form of papers generated during the creation of the thesis are listed. Finally, the document structured is presented.

1.1 New Mechanisms for Programming Robots

Robots are finding their way out of their cages. In industrial environments, robots working in production lines, usually called industrial robots, are confined to metallic cages, where they spend their workday. This cage is built to protect workers from a possible collision with the robot, which may cause severe injuries to the worker. Robots have been used this way for a long time. With the introduction of Unimate in 1961 (Mickle, 1961), the industrial robotic age has continued to our days.

However, with time comes evolution, and robots have been redesigned to be able to interact with humans in a non-dangerous way. This evolution has allowed robots to be programmed using new mechanisms. One way to program these robots is through imitation. Inspired by how humans can learn new tasks, roboticists have created mechanisms in which humans can demonstrate robots how to perform a task. This is called *robot imitation*. Among the most used paradigms inside robot imitation is Programming by Demonstration (PbD).

In classical PbD, the robot perceives a human performing an action. After several demonstrations of the action, the robot is able to create a kinematic model. This is

called generalization, because it generalizes from a set of similar but slightly different demonstrations. This kinematic model is a representation of how human limbs and body have moved during the task.

One problem comes when transferring these parameters to the robot. Even in humanoid robots, which resemble a human form, kinematic configuration discrepancies between learner and demonstrator are noticeable: a longer or shorter arm, a different range in elbow joint, different thigh weight, different height, different hand size, etc. This adaptation from the human kinematics realm to the robot kinematics realm is called the *correspondence problem*. In one variant of PbD, the human physically moves the robot, guiding it during the tasks completion. This is called kinesthetic teaching. The generalization, in this case, is directly modeled in the robot kinematics realm.

Whatever the variant of PbD chosen, this paradigm is kinematics-focused, which means that only kinematic parameters are modeled (joint angles, end effector positions, velocities, etc.). This fact limits the flexibility and applicability of robots to complete tasks. Robots are unaware of the task objectives when performing it, they simply execute a series of numeric commands in order (e.g. joint i to 30 degrees).

A more useful approach to imitation is goal-directed imitation. In this case, instead of modeling the kinematic parameters of the task, the effects of the action are modeled. For example, imagine a task where the objective is to close a window in a room of a house. We have two options: To teach the robot how to close the window from every position and orientation, with both hands, or to close the window and let the robot model the changes in the room and reproduce the action on its way. Which approach should be considered more useful?

1.2 Overview and Definitions

The problem this thesis aims to solve is the imitation of actions based on goals. These goals are the changes introduced in the environment by the action. Specifically, actions where the changes produced after the task, but also during it, are taken into account. A high level representation of this CGDA core idea is depicted in Figure 1.1.

Throughout the thesis, nomenclature from robot imitation will be used. Despite the lack of standardization, commonly used terms can be listed. Also some definitions are given:

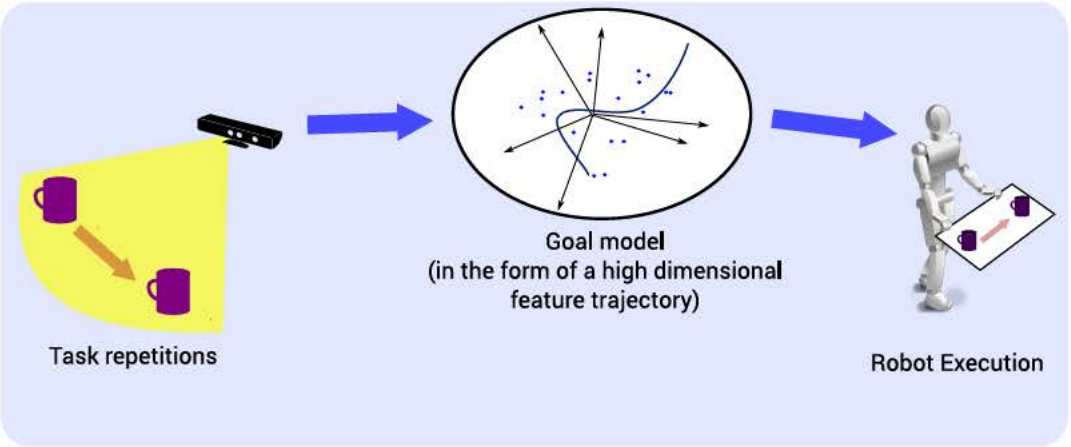

Figure 1.1: *Continuous Goal-Directed Actions sketch: From a set task demonstrations (also called repetitions), a goal model is created. Then, action reproduction (also called execution) can be performed.*

- **Actions**, **Tasks**: A piece of work done as part of robot's duties. Examples: painting a wall, filling a glass with water, moving objects in a room.

- **Repetitions**, **Demonstrations**: A human repeats a task for the robot several times.

- **Objectives**, **Aims**, **Goals**, **Targets**: The effect of an action which motivates its reproduction. For example: the goal of painting a wall is to obtain the effect of the wall changing its color.

- **Environment**: Despite a more generic definition of environment is accepted by the paradigm developed in this thesis, in our work, environment is defined by the objects affected by the action. These objects are usually in the form of discrete objects on tables (mugs, pencils, water glasses, etc.) or in the form of room elements (walls).

- **Learner** (robot): The robot recording the repetitions. This robot is also supposed to reproduce the learned actions, but it is not compulsory. Any robotic shape is allowed as long as it has sensing capabilities: video cameras, force sensors, microphones, etc. In our case, it is a humanoid robot.

- **Demonstrator**, **Teacher** (human): The human person demonstrating the action to the robot.

- **Generalization**, sometimes **Learning**: The model the robot extracts from the set of demonstrations. It may a be a kinematic model or a goal model.
- **Comparison**, **Recognition**: Barely covered explicitly in literature, but implicit in most frameworks, it is used to measure how similar two models are.
- **Execution**, **Reproduction**: The accomplishment of the task by the robot. The robot moves to complete the previously modeled task.
- **Learning from Demonstration (LfD)**, **Programming by Demonstration (PbD)**, **Imitation Learning**, **Apprenticeship Learning**: Paradigm for enabling robots to autonomously perform new tasks based on human demonstrations. Usually kinematic parameters, but also dynamic ones, are the raw material for analysis. Prototypical blocks are: Data acquisition (sense), Generalization (plan) and Reproduction (act).
- **Goal-directed actions**: Paradigm where the only parameters analyzed are the ones belonging to the elements affected by the action.
- **Goal-only actions**: Goal-directed actions where the only information used is the difference between the initial and the final state of the environment.

With the information provided above, let us extract a formal definition of the paradigm created in this thesis.

Definition 1 ***Continuous Goal-Directed Actions*** *is a paradigm of robot imitation based on goals through time. In this paradigm, the continuous temporal state of the environment, affected by the action, is recorded and modeled. In the model created, each feature recorded is a dimension in a high dimensional feature space. An action inside this feature space is represented by a trajectory.*

1.3 Motivation

In the past, robots were preprogrammed. The environment was meticulously defined for the robot, and its tasks were constant and repetitive. In the current age, industrial robotics systems, specially for manufacturing processes, are taking advantage of Programming by Demonstration paradigm. In this paradigm, a user can guide the

robots through the path the robot will follow later on its own. It is reasonable to think that in the future, robots will be indicated the task to perform, by voice or gestures, and the robot will understand how to complete the task. User may also play a teaching role by demonstrating the action to the robot. A target scenario can be found in Figure 1.2.

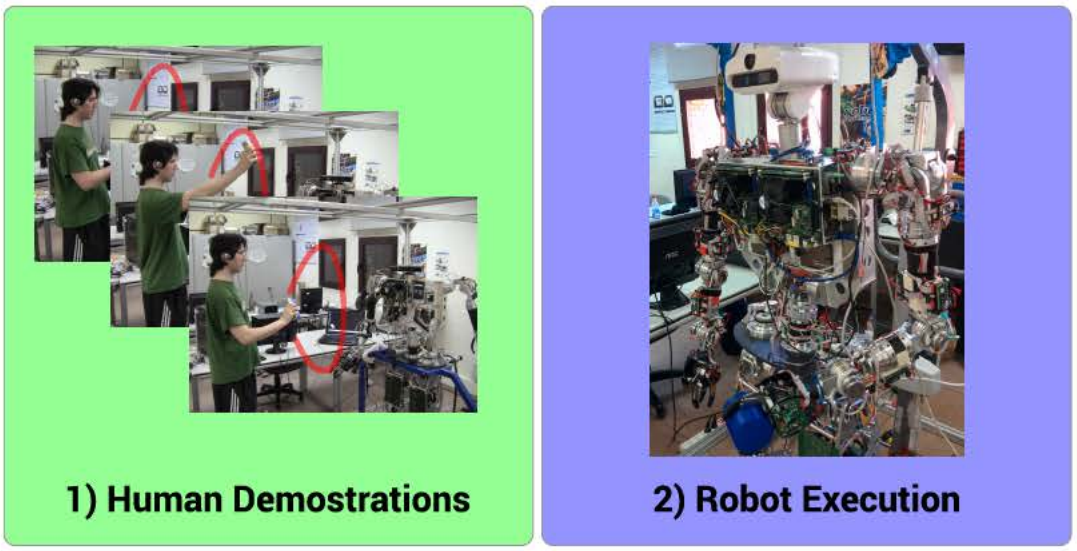

Figure 1.2: *This figure represents the beginning and the end of an imitation scenario. It starts with a series of task repetitions perform by a human and recorded by the robot. It ends with the robot reproducing the task by itself.*

It is in this context of goal-directed actions this thesis is developed. Some problems appeared when attempting to perform goal-directed actions with robots.

1. Robots do not understand the effects of actions.
2. Robots are also unaware of the differences between actions, or when two actions have similar goals.
3. If the user provides a set of demonstration of a task, the set may include bogus ones. In addition, every feature measurable by the robot's sensor will not be relevant for the task, which may distort the perceived goal.
4. Robots do not know how to perform goal-directed actions. They must be provided a kinematic description to perform the tasks. Even with a good task description, in a goal-directed sense, the robot must discover which of its movement returns the desired output.

5. Kinesthetic teaching has increased the necessity to improve the existing physical interaction mechanisms (user-easy, robot-safe). Robot have heavy mechanical links and electric motors attached to gearboxes that introduce high frictions. Trying to physically guide the robot is a tedious and difficult task.

This thesis aims at solving the exposed problems.

1.4 Main Objectives

Considering the previously presented problems, this thesis proposes the following solutions.

1. To provide a framework able to measure and generalize effects of actions on the environment.
2. To create metrics to compare goal-directed actions.
3. To develop techniques enabling filtering bogus demonstrations and unnecessary features.
4. To generate motor primitives using real robots, assuring the robot can perform these movements.
5. To facilitate the creation of these motor primitives by partially compensating friction and gravity.

Experiments has been designed for each research topic presented. These experiments focus on testing the decisions made to build the framework.

1.5 Scientific Contributions

This thesis has been built using several research publications as scaffolds. The research performed during the development of this thesis led to the publication of 6 directly related papers in international peer-reviewed conferences, symposiums and scientific journals.

The Continuous Goal-Directed Actions theoretical paradigm and practical framework was sketched in (Morante, Victores, Jardón, & Balaguer, 2014) and expanded in

(Morante, Victores, Jardón, & Balaguer, 2015). The machine learning algorithm that allows the automatic demonstration and feature selection was presented in (Morante, Victores, & Balaguer, 2015a). The creation of Guided Motor Primitives was developed in (Morante, Victores, Jardón, & Balaguer, 2014). To facilitate the creation of these primitives, friction and gravity compensation controllers were added in (Morante, Victores, Martinez de la casa, & Balaguer, 2015). Finally, the brief analysis of the cyber security of robots was published in (Morante, Victores, & Balaguer, 2015b).

The thesis code[1], its associated tools, experiments, results, and slides are publicly available, and have been open-sourced.

1.6 Document Structure

The document structure is presented in this section.

- Chapter 1 is the introduction to the thesis, and contains a brief history of mechanisms for programming robots, the thesis overview and terms definitions, the motivation to follow this line of research, the main objectives this thesis aim to reach, the scientific contributions generated, and this document structure.
- Chapter 2 presents the state of the art of each research topic.
- Chapter 3 provides the introduction and general description of the Continuous Goal-Directed Actions framework.
- Chapter 4 is dedicated to filter the bogus demonstrations performed by the user and also discard those features not relevant for the task.
- Chapter 5 deals with generalization, also called model construction, and the metrics to perform comparisons among models, also called recognition.
- Chapter 6 focuses on the execution of actions encoded as goal-directed actions. It also includes the creation of motor primitives and the necessary controllers to alleviate this operation.
- Chapter 7 presents the experiment validation of the research developed in previous chapters.

[1] https://github.com/smorante/continuous-goal-directed-actions

- Chapter 8 discusses the limitations, shortcomings, potential features, strengths and future works of the thesis. It also depicts the conclusions.
- Appendix A briefly analyses the cyber security of current robotic platform, and their software components.
- Appendix B attaches general information about TEO the humanoid robot and its physical characteristics.

Chapter 2

Background

This thesis encompasses several different topics, composing a relatively large framework. Every topic application is analyzed throughout the thesis, and each of them deserves a background section of their own, for a better understanding of the thesis.

2.1 Programming by Demonstration

The field of robot imitation has been dominated by motor parameter reproduction (A. Billard, Epars, Calinon, Schaal, & Cheng, 2004). This approach has been called *Programming by Demonstration* (PbD) (Calinon, D'halluin, Sauser, Caldwell, & Billard, 2010) or *Learning from Demonstration* (LfD). These methods encode an action by recording the joint motor parameters of a demonstrator when performing the action, and then applying different machine learning techniques to extract a generalization. The demonstrator can either be the guided robot itself, or an external agent. A high-level overview of this paradigm is shown in Figure 2.1. In this case, the kinematic parameter of a human demonstrator is used as the raw data to analyze.

Let us overview some representative examples of PbD. In (Calinon et al., 2010), a human demonstrator performs a task several times (e.g. hitting a ball) using a robotic arm. Positions, orientations and velocities of the arm are recorded (see Figure 2.2). The number of representative states of the action are estimated with Hidden Markov Models (HMM). HMM are used to handle spatio-temporal variabilities of trajectories across several demonstrations. Finally, and in order for the robot to execute the trajectory, Gaussian Mixture Regression (GMR) is used to create a regression function using

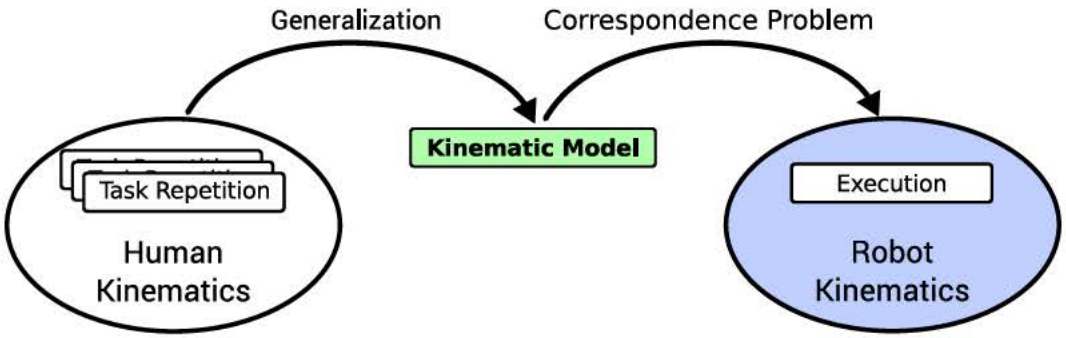

Figure 2.1: *The framework usually starts with a human performing several demonstration of the task. From this point, a generalization process generates a task model, in human joint terms. Then, the correspondence problem is solved, to some extent, to translate the task into the robot morphology. Finally, the task can be reproduced in robot joint terms.*

previous HMM states. This reconstructed trajectory is the one the robot reproduces to imitate the human movement. Another common technique used, along with HMM (Calinon & Billard, 2004) (Calinon & Billard, 2005), is Gaussian Mixture Models (GMM) as in (Calinon & Billard, 2007) and (Calinon, Guenter, & Billard, 2007).

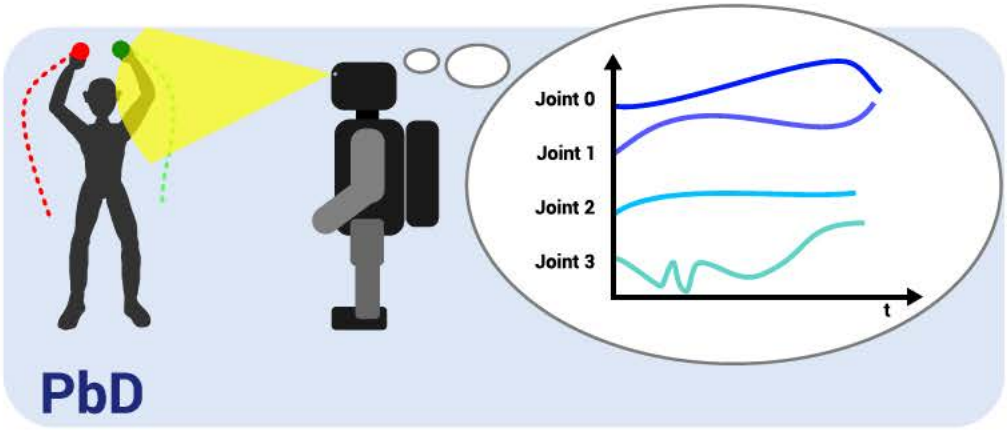

Figure 2.2: *PbD starts with a human performing a task. Kinematic parameters are recorded for a later generalization.*

Some authors questioned whether this motor imitation alone is useful for intelligent robots. Schaal asked in (Schaal, 1999) as an 'outstanding question' to be answered: *How can the intention of a demonstrated movement be recognized and converted to the imitator's goal?*. From a wider perspective, (Nehaniv & Dautenhahn, 1999) summarized the three key concepts of imitation: 'what to imitate', 'how to imitate', and 'when to imitate'. The

Continuous Goal-Directed Actions encoding of tasks aims to fulfil mainly the 'what to imitate' concept (Morante, Victores, Jardón, & Balaguer, 2014).

The author believes that robot imitation could be improved, and some of the problems stated solved, by taking more into account the action consequences in the environment. These consequences will usually be the goals of the task ('paint' action modifies the *color* of the painted object).

Recognizing an action through external measurements is called direct action recognition. In (Subramanian & Suresh, 2012), they perform a neuro-fuzzy classification of optical flow features between consecutive frames of human movement in video sequences. Neuro-fuzzy is a combination of fuzzy logic with neural networks, using the classified output of a fuzzy system as an input to the neural network. In (Chivers, 2012), they track and filter human hand and feet trajectories through Principal Component Analysis (PCA). First, they record trajectories of key points from a video. Then, they split them into sub-units called basic motions. Next, they extract some features of the basic motions, and project these feature vectors into a reduced space generated by PCA, resulting in the formation of clusters of similar actions. For recognition purposes, they record an action, transform it with the same process explained, project its vector onto the reduced space, and finally, associate it with the closest cluster.

As mentioned, the focus in these types of research is on learning the kinematics of actions. By using only kinematics, actions are limited to be executed exactly as taught. Any disturbance, e.g. a blocking path or a displaced element, would make the task completion impossible. This is why a complementary effect encoding is also important. Goals can give a meaning to the task. It is the focus of this thesis to study goal-directed actions, and it is also the main topic of the next section.

2.2 Goal-Directed Actions

Humans are able to easily extract the main consequences of an action performed on an object. However, in usual robot imitation, there is a lack of codification of action effects, and only the kinematic aspects are considered (it is a kind of blind imitation). This fact limits flexibility in action execution. A goal-directed framework would flow as in Figure 2.3. In this case, the generalization is based on goals, and the demonstrator kinematic parameters are not taken into account.

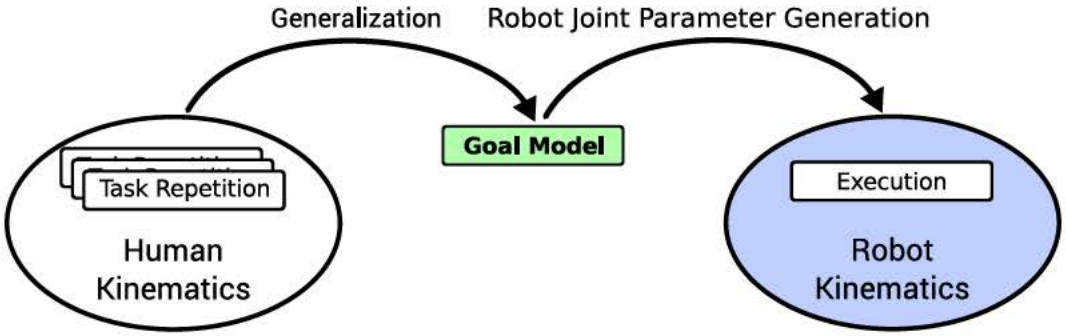

Figure 2.3: *The goal-directed framework usually starts with a human performing several demonstration of the task. From this point, a generalization process generates a goal model, in terms of the consequences of the action in the environment. Then, the execution problem is solved to translate the goal model into a kinematic model. Finally, the task can be reproduced in robot joint terms.*

When looking at how nature faces the problem of imitation, psychology indicates that the human brain encodes actions as end-goals. For example, when children imitate others grasping a person's ear, they tend to imitate the action goal (which ear to grasp) rather than the kinematic aspects of the action (which hand is used to perform the grasping) (Bekkering, Wohlschlager, & Gattis, 2000). However, in usual robot imitation, there is a lack of codification of action effects, and only the kinematic aspects are considered. This fact limits flexibility in action execution. The robot can, effectively, repeat the same movements, but the action goals remain a mystery for it (Nehaniv & Dautenhahn, 1999).

Neuroscience has also discovered some points supporting goal coding of actions, especially in the study of *mirror* neurons (Gallese, Fadiga, Fogassi, & Rizzolatti, 1996). This behavior seems to be related with the mirror neuron system (Rizzolatti, Fadiga, Gallese, & Fogassi, 1996). Mirror neurons fire both when an action is performed and when the same action performed by another subject is observed. As shown in (Rochat et al., 2010), when monkeys were trained to grasp food with a tool, their mirror neurons activated when they performed the action, and also when observing the action performed by an experimenter using a different tool. This demonstrates their kinematic independence.

Inspired by this neuronal behavior, this thesis present Continuous Goal-Directed Actions (CGDA) and define them as actions in which the only parameters analyzed are the ones belonging to the object (or more generally, elements) affected by the action

in a continuous time. As defined previously, Goal-directed actions can be splited into Goal-only (like in (Victores, Morante, Jardón, & Balaguer, 2013a)) and Continuous Goal-Directed Actions (like in this thesis). An example of their differences can be seen in Figure 2.4.

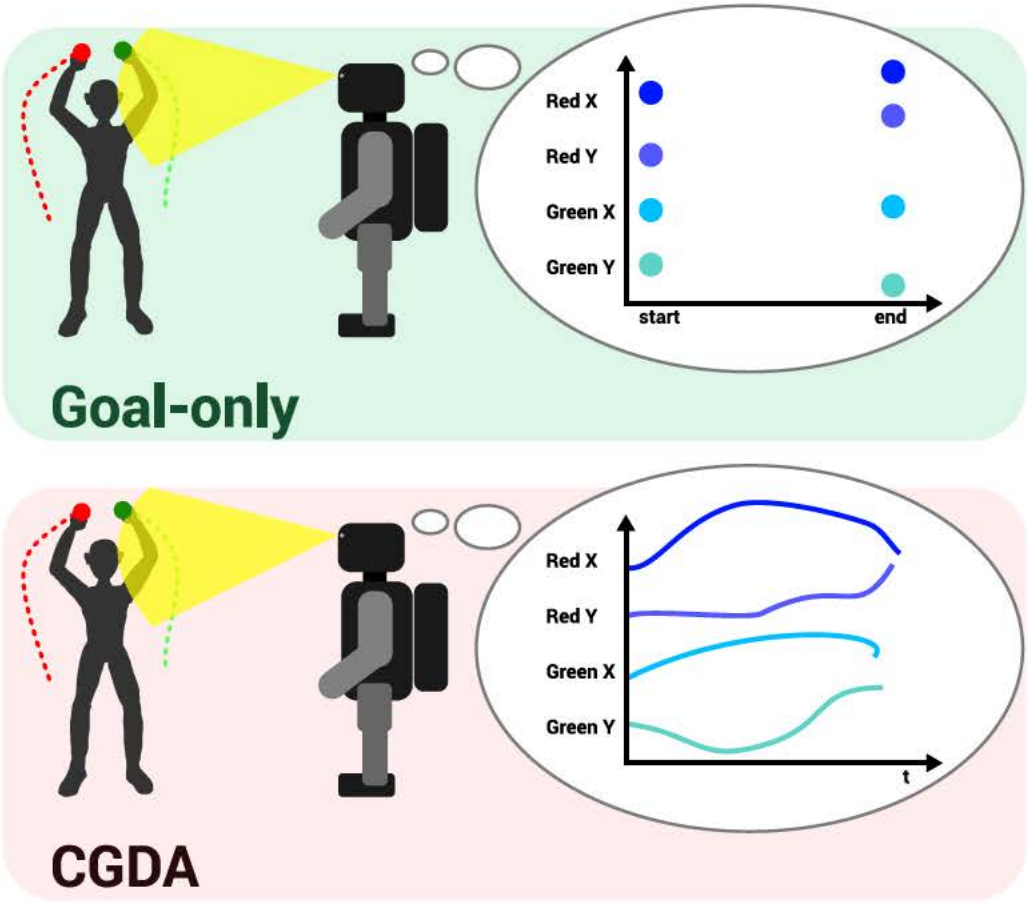

Figure 2.4: *Goal imitation starts with a human performing a task. Object features are recorded for a later generalization. In the goal-only case, only the start and the end state are recorded. In CGDA case, the full sequence of states are recorded.*

Continuous Goal-Directed Actions seems, by its definition, a useful way to teach robots the consequences of an action. Using CGDA, an action can be encoded in a complementary way, by learning, not only how to do it in terms of kinematics, but also the effects of the action. This *double learning* may allow the robot to complete the action, even when the scenario changes or when elements block its usual path of execution, by generating alternative motions which accomplish the encoded goals. The prototypical

example of why it is convenient to move from a goal-only paradigm to a Continuous Goal-Directed one is a valve. Imagine a task consisting in rotating a valve one complete revolution of 360 degrees. A goal-only encoding would not capture any difference between the initial an the final state. However, CGDA would record the whole process of turning the valve, understanding the task's objective.

When talking about goal-directed actions in robots, a goal encoding is found in (A. Billard et al., 2004). They extract goals as relevant features that appear most frequently from a demonstrated dataset. The goals are those invariants in time. This framework was extended in (Calinon, Guenter, & Billard, 2005a). Despite they learn the kinematic trajectory to perform actions, they encode action goals. They were replicating a psychological experiment with children (Bekkering et al., 2000). The setup is the following: on a table, there are colored dots which are touched by a human with both arms in alternation. When the dots stay on the table, children tend to imitate the goal (what dot to touch), and not the arm used to do it. In the robotic experiment, during the demonstration, the robot tries to extract a set of invariant constants. Later, the robot computes the trajectory that best satisfies the constraints and generates a motion.

Another example is (Erlhagen et al., 2006), where an object must be grasped and then placed at one of two presented targets that have different heights. There is a bridge shaped obstacle in the path. Depending on the height of the bridge, the object must be grasped differently and through a different path. In (Saegusa, Metta, Sandini, & Natale, 2013) there is a learning phase where the robot generates motion (as a combination of motor primitives) with no specific purpose, and analyses the consequences (sensory effect) of its actions. After this learning phase, when a demonstrator performs actions, the robot recognizes the observed actions by observing the consequences and encoding them as combination of motor primitives.

Unfortunately, there are no exclusively Continuous Goal-Directed Actions references in literature, to the author's knowledge. A relatively close work (Johnson & Demiris, 2004), which uses a combination of object spatial and demonstrator hand movement tracking. They build a system with a set of primitive actions (inverse models). When a human demonstrator performs an action, they continuously track the object and the demonstrator's hand spatially through time. At the same time, they run all inverse models during action stages to find the best performance of each model in each stage. Finally, they construct a high-level inverse model composed by the selected

primitives, being able to imitate the action goal with similar spatial movements. The object tracking is only used to identify grasping and releasing stages. Despite the continuous tracking used in this work, they do not fully exploit the benefits of the object features variation.

Tani's group developed an inverse model by training a multiple timescale recurrent neural network to match robot motor torques with the position of an object in (Yamashita & Tani, 2008). Similarly, vision is only used to track the spatial position of the object.

Goal-directed actions are interesting because they enable a robot configuration independent way to encode and execute actions. Goal-directed actions, instead, assume internal mechanisms to perform a demonstrated action by indicating the desired goals. This allows the system to avoid the correspondence problem, which has its origins in the difference in the kinematic model of the demonstrator and the learner (Mohammad & Nishida, 2013).

One requirement in any robot learning paradigm is that the information provided to the robot must be relevant to generalize an adequate model. However, it is difficult to know a priori which features will be relevant. It is also important to perform repetitions correctly, as improper demonstrations will lead to poor models. During the development of CGDA, we realized these problems, and developed demonstration and feature selection techniques to avoid it. The next section will cover this topic.

2.3 Demonstration and Feature Selection

One way to teach modern robots is to perform the desired task several times, while the robot records data. The robot is supposed to be able to generalize one single adequate task. However, not all the demonstration presented by the user may be correct, and not all the features may be relevant for a particular task. It is necessary to screen both the demonstrations and the features (represented in Figure 2.5).

The general problem of selection of demonstrations and features, for the robot learning scenario, can be decomposed into three subparts: different duration of user demonstrations, demonstration selection, and feature selection.

One of the problems that arises when comparing user demonstrations is their different duration. It is tricky to compare sets of multidimensional signals of different du-

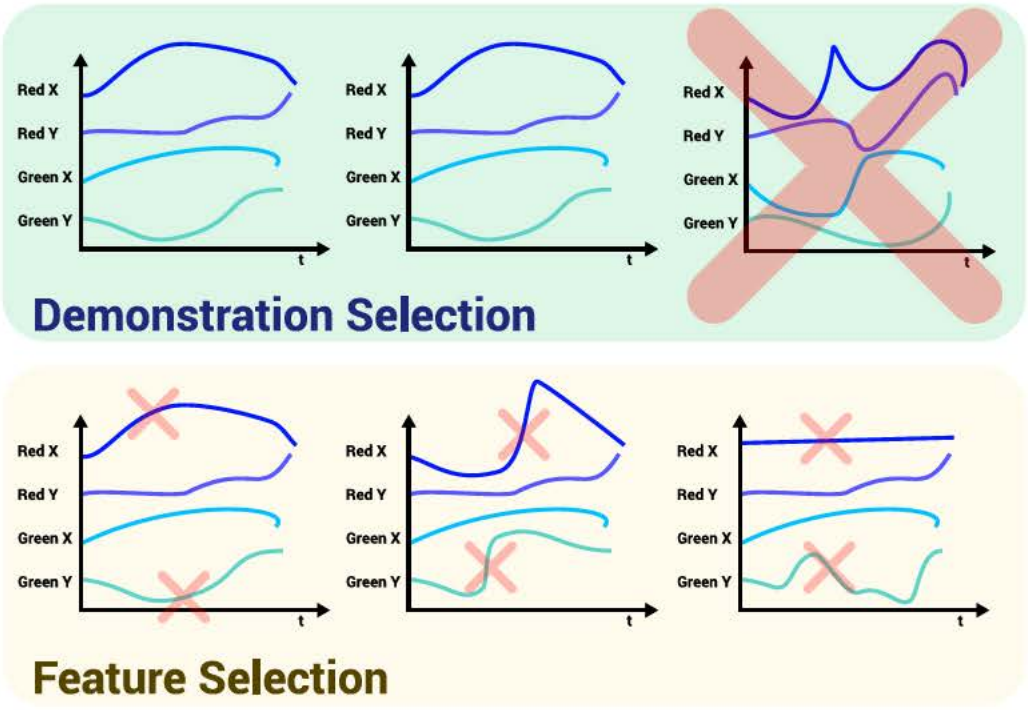

Figure 2.5: *In the upper block, demonstration selection is carried out. From the set of complete demonstrations (each one represented as an axis frame with blue lines), the most different are discarded. In the lower block, feature selection is carried out. From the set of features, those feature which are not similar across demonstration are removed.*

rations without losing too much information. Scaling and warping can lead to desynchronizing demonstrations and feature trajectories within demonstrations. Literature has faced this problem in different ways.

Some machine learning authors force the user to perform all the demonstrations with a fixed duration (Atkeson & Schaal, 1997). Jetchev also fixes the duration (Jetchev & Toussaint, 2011) but claims that different duration demonstrations can also be handled. Mühlig uses Dynamic Time Warping (DTW) on each individual feature when comparing the signals, in his works (Muhlig, Gienger, Steil, & Goerick, 2009) and (Muhlig, Gienger, Hellbach, Steil, & Goerick, 2009).

Statisticians face similar problems when comparing temporal series. Some techniques they use are (Tarsitano, 2012): padding (filling the shortest signal with zeros at the end), upsampling (inserting values in the middle of the signal), or interpolating (interpolate timestamps to the range of the largest signal). These options introduce

artifacts that distort the information contained in the signals.

When recording user demonstrations, there is no guarantee that all the repetitions will be perfectly recorded or executed by the demonstrator. There may be many reasons for this: human fallibility, sensor error, network latency, etc. Sometimes it is difficult and time-consuming to manually check each demonstration and each feature to find anomalies. Additionally, when recording many features at a high rate sampling, the data generated can easily overwhelm the human capacity to find deviations from the correct demonstration. Chernova (Chernova & Veloso, 2008) proposes a method to filter discrete choices in a human-robot interaction reinforcement learning framework. However, to the author's knowledge, there has not been any work on automatic demonstration selection, where incorrectly performed or sensed complete user demonstrations are discarded.

Many possible features may be extracted from sensor data for each task, but not all of them may be relevant for a specific task. A feature selector can automatically discard features that are irrelevant for a given task. While most robot learning frameworks are provided only the relevant features considered by their designer (i.e. only joint angle values or operational space coordinates), in certain literature more features are fed to the algorithms, some of which are automatically selected.

In (Calinon, Guenter, & Billard, 2005b), the features used for encoding the task are the robot joint angles, the user hand coordinates, the location of the objects at which actions are directed, and the laterality of the motion (which hand is used). They encode the trajectories into a Hidden Markov Model (HMM) of the task for each demonstration. For feature selection, they discard features that present a high variance among HMM states.

Variance is also the discarding factor in (Muhlig, Gienger, Hellbach, et al., 2009), where the observed movement is projected into a task-specific space and the correspondence problem is avoided by solely focussing on the object trajectories without making any assumption on the teacher's postures during the demonstration. They encode relative object positions and orientation. They use Dynamic Time Warping (DTW) to avoid the different duration problem, and they discard features by variance.

The same author extended the discarding possibilities by adding an attentional factor or a energy saving (called kinetic) factor (Muhlig, Gienger, Steil, & Goerick, 2009). They use what they call task spaces. Observed movements are mapped into a pool of

task spaces and they present methods that analyze this task space pool in order to acquire task space descriptors. A selection method named task space selector analyzes the observed object trajectories and acquires task space descriptors that match the observation best. Several criteria are incorporated, such as a psychologically inspired criterion that is based on the robot's attention to the objects in the scene and a kinetic criterion that estimates effort and discomfort of the human teacher. Concerning the learning of object movements, task spaces may be composed of absolute object positions and orientations, relations between objects, additional constraints such as the restraint to only planar movements, and additional joint-level constraints.

In (Jetchev & Toussaint, 2011) they encode the center of the target object, three fingertips, the three lower digits, the palm center and the relative distances. They remove the redundant features using correlation as a measure.

With a good selection of algorithms for demonstration and feature selection, relevant information is obtained, which can lead to better models. With these models, the robot has to be able to reproduce the action by its own. This is a non-trivial challenge, as the space of movements to search is large and high-dimensional. One method to reduce the search space is to constraint to look for movements we know a priori the robot can execute. If these movements are small basic motions they are called *motor primitives*. Next section will deep in the topic.

2.4 Motor Primitives

Literature has provided insights on how the human brain may use motor primitives for performing complex actions (Schaal, Ijspeert, & Billard, 2003). In his influential paper (Schaal, 1999), Schaal describes the area of movement primitives what he defines as *sequences of action that accomplish a complete goal-directed behavior*. The development presented in this thesis is close to this definition, despite the related works in motor primitives sometimes do not focus enough on the word "sequences". Instead of generating a single movement primitive to encode complete temporal behaviors, this thesis aims to split movements and create small basic motions, which may be able to complete the required task when combined sequentially, like in Figure 2.6. This is an attempt to answer other Schaal's 'outstanding questions': *Is there a basic set of primitives that can initialize imitation learning?, How complex are the most elementary primitives in this set?,*

How can new primitives be learned?, How is sequencing and the recognition of sequences of movement primitives accomplished?

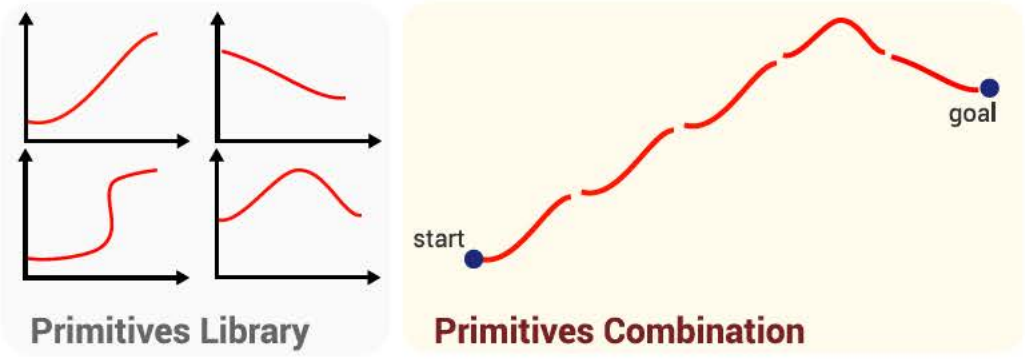

Figure 2.6: *From a set of primitives, a combination which accomplishes the goal is extracted.*

Dynamical system Movement Primitives (DMP) is one of the most famous primitive frameworks in robot imitation. They were introduced in (Ijspeert, Nakanishi, & Schaal, 2002) and were used in many subsequent works (e.g. (Kober, Wilhelm, Oztop, & Peters, 2012), (Peters, Kober, Mülling, Krämer, & Neumann, 2013)). In DMP, dynamical systems are used to encode a policy. A policy is a mapping between world state and actions which enables a robot to select an action based upon its current world state (Argall, Chernova, Veloso, & Browning, 2009). This policy serves to accomplish an action by obtaining a reusable parametrized single-primitive policy. DMP have been successfully used for learning complex, but unitary, tasks, like learning single primitive actions (Peters & Schaal, 2008) (Kober, Mohler, & Peters, 2008). Examples of these unitary tasks include: hitting a ball in a ping pong game, filling a glass with water, putting objects on a table or clean a blackboard.

Another approach can be found in (Mülling, Kober, Kroemer, & Peters, 2013). They aim to acquire a library of primitives and select among them to adapt to new situations. For this adaptation they use primitive-associated parameters (called augmented state) which describe the task context from which the primitive was extracted. They combine weighted primitives to generate a movement. The demonstration set is obtained by guiding the robot to perform the desired tasks.

A developmental approach for self-discovery of primitives is found in (Ugur, Sahin, & Oztop, 2012). Similar to (Mülling et al., 2013), they encode primitives with descriptors, which describe situational parameters. For self-discovering motor primitives, they

first execute several actions which randomly combine different joint speeds and grasp states of the robot hand. For extracting generic behaviors, the system finds segments with the same initial-end situation. These segments are grouped and combined into a single representative behavior primitive, computed by taking the average of initial and final velocities. As in previously mentioned papers, this work only considers single primitives, and sequential combination of motor primitives is not studied.

There are several ways to create a library of primitives e.g. pre-programing or motor babbling. One very popular way to create primitives is to physically move the robot. While this seems a comfortable and fast approach, the mechanical construction of robots makes the movements tough and the user must also deal with joint stiffness. In this thesis we aim to create a very simple partial friction and gravity compensator to assist in Programming by Demonstration. The next section will expand on this.

2.5 Physical Interaction with Robots

Many robots, specially humanoid robots, are stiff in their movements. They are built with heavy metallic mechanical links and electric motors attached to gearboxes that introduce high frictions. This fact makes it very difficult to physically interact with the robot. With the advent of paradigms such as Programming by Demonstration (PbD) (Calinon et al., 2010), where physical movements are used to program the robot, there has been an increasing necessity to improve the existing physical interaction mechanisms.

It is not this thesis' objective to develop a complex friction model. Literature in this topic is very broad and specialized. This work solely aims to aid at guiding robots in their movements. Any compensation, even a partial one, improves the performance of the guiding.

In kinesthetic teaching, a popular choice in PbD, the robot's motors are set to a passive mode where each limb can be driven by the human demonstrator (A. G. Billard, Calinon, & Guenter, 2006). Some authors suggest that kinesthetic demonstrations are more intuitive for naïve users, but that this fact changes when facing with high degree of freedom (DoF) robots (Akgun, Cakmak, Yoo, & Thomaz, 2012). Akgun et al. present an alternative, called keyframe demonstration, where key positions of the task are recorded, while the intermediate movements are interpolated. For instance, Bax-

ter robot uses this technique of recording frames to be programmed, aided by gravity compensation (Guizzo & Ackerman, 2012). While gravity compensation is useful for providing kinesthesic demonstrations, and additional friction compensation term may aid in keyframe demonstration. Simpler interactions with robots are possible, if the robot moves in the direction indicated by applying small forces (Figure 2.7 depicts this objective).

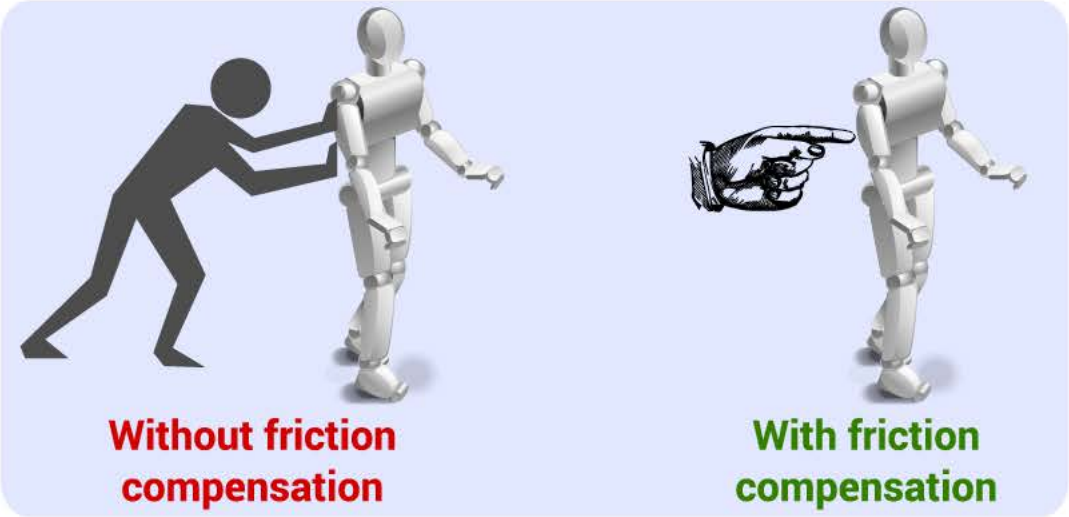

Figure 2.7: *Without compensation, robots are very stiff to be moved. With compensation, even a partial one, robots can be guided easily.*

.

The main fields of study for the development of this thesis' compensation controllers are related with friction and gravity compensation. Only selected background works will be mentioned, as the literature in friction compensation is extensive. A more in-depth review can be found in (Olsson, Åström, Canudas de Wit, Gäfvert, & Lischinsky, 1998).

On one side, friction is described as the resistance of motion of two contacting sliding surfaces (Olsson et al., 1998). To measure friction accurately is extremely difficult. Exact models of friction do not exist, and instead approximations obtained through experiments are used. Proposed models include: Coulomb, viscous friction, Stribeck, Dahl, LuGre, Leuven, etc. No specific model has proven better than others (Bona & Indri, 2005).

Canudas et al. (Canudas, Astrom, & Braun, 1987) focused on modeling non-linear

effects of friction in DC motor drives. They combine a linear model for viscous friction with a parameter estimation algorithm, which recalculates linear model parameters in a feedback loop to reduce the error in velocity commands. Some methods for friction identification in robotics consider elements in isolation, or do not consider mechanical limitations (Kostic, de Jager, Steinbuch, & Hensen, 2004)(Papadopoulos & Chasparis, 2004). A low-velocity approach allows obtaining friction models depreciating inertia in (Kermani, Wong, Patel, Moallem, & Ostojic, 2004). As modeling motor frictions involves non-linearities (Stribeck effect, hysteresis, pre-sliding displacement, etc.), some authors (Na, Chen, Ren, & Guo, 2014) have delegated this problem to learning algorithms such as Neural Networks. Gearboxes also have high frictions, and additionally increase motor frictions from the link's point of view (due to the reduction factor).

The most popular gearboxes in humanoid robotic platforms are Harmonic Drives, because of their compactness and reduction factor. Authors (Gomes & Santos da Rosa, 2003) have tried to model Harmonic Drives' frictions, finding similar problems of non-linearities as those of the motor case. Regarding humanoid robots, in (Traversaro, Del Prete, Muradore, Natale, & Nori, 2013) they identify friction parameters on an iCub robot, aided by 6-axis force/torque sensors.

On the other side, gravity compensation is computed using the dynamic model of the robot. By analyzing the kinematic configuration and the masses of links and motors, it is possible to calculate the influence of gravity in each motor, and compute the torque value necessary to compensate it. In (Luo, Yi, & Perng, 2011) they compensate gravity by projecting gravity forces on each joint of a robot arm. First, they translate all joint coordinates to the base frame. Then, they project on each joint, the torque generates by gravity forces on the rest of links and motors. This method is a simple and methodical procedure to compensate gravity in rigid links.

In classical literature, the inclusion of a gravity compensation term in robot manipulation control schemes was used for improving a PD position control (An, Atkeson, & Hollerbach, 1988). Including gravity compensation performed as well as a full feed-forward controller with full inertial terms. Another work (Liu & Quach, 2001) aimed at estimating and compensating gravity and friction forces in the context of improving the position error in robot manipulators. However, the possibility of simulating free movements was not studied. Also, the use-case of these works in usually to improve the position control in industrial robots. Our use-case is aiding at Programming by

Demonstration, which has different assumptions and targets.

2.6 Chapter Summary

In this chapter, we have covered the state of the art of the main topics of the thesis. These topics are: robot imitation, with an emphasis on Programming by Demonstration, goal-directed actions, techniques for demonstration and feature selection, the creation of motor primitives, and friction and gravity compensation for physical interaction with robots.

In the next chapter we start describing the original works developed. We start by defining the core of CGDA, and continue by explaining each component surrounding the core.

Chapter 3

Continuous Goal-Directed Actions

Continuous Goal-Directed Actions (CGDA) is a framework to encode the effects of an action when the action is demonstrated to a robot. We have developed this framework to allow the learning of actions with relevant object feature intermediate states e.g. recognizing the rotation of a valve is unachievable without a continuous tracking, because the final state of the valve could be the same as the initial, looking like no action has been executed. Let us state some advantages over similar paradigms. **Advantages over Programming by Demonstration**:

- **CGDA captures the objective of the action**, beyond the kinematic movements to accomplished the task.
- **CGDA can transfer actions between robots seamlessly**, while PbD must solve the correspondence problem.
- **CGDA can compare new actions with previously seen ones**. It can compare its goals, and if they are similar enough, it could reuse behaviors used in previous tasks. PbD can be applied analogously, but only if the actions have similar kinematics, which may not be case. For example, grasping and moving a glass with a handle is kinematically different than moving a glass without handle.

Advantages over goal-only actions:

- **CGDA can reach intermediate goals in actions**, because it records the whole task features. For example, any tasks involving object rotations (knobs, valves) are unachievable without taking into account intermediate states.

Pick and place actions are the scenario where goal-only actions succeed. If the only target is to move an object from A to B, a goal-only encoding is an optimal way to do it. Once the task has been generalized, the robot has to simply execute a predefined *move* behavior to complete the task. Nevertheless, picking an object, using it, and returning it to its original place could be challenging for goal-only encodings. Moving from A to B, using the object, and returning it from B to A does not fit in the start-end encoding.

Other examples where CGDA may succeed but goal-only cannot: picking a lighter from a drawer, light the stove, and return it to the drawer; turning the lights on to pick a mug, and switch them off to leave the room; screwing a screw; playing an instrument, etc.

A critic person could argument that a solution to this problem could be to split the task in smaller sub-tasks with their own target and tackle them in order. This is precisely what CGDA does: **CGDA considers each timestep a sub-tasks to be reached in order**.

- **CGDA simplifies learning complex tasks**. Imagine a task consisting in painting a wall of blue color. Goal-only encode would record the initial state (white wall) and the final one (blue wall). Theoretically, goal-only could generate behaviors to paint the wall, but the search space is massive. However, if intermediate states are recorded (e.g. small wall areas change their color), the search space reduces, as you can try to reach each simple "waypoint" in order.

A complete block diagram of the research carried in this thesis can be found in Figure 3.1. In this chapter we will discuss the differences between CGDA and Programming by Demonstration. We will also briefly overview each CGDA component developed in the thesis. In-depth analysis of each component will be included in their correspondent chapters.

3.1 Differences between CGDA and PbD

Robot imitation is usually performed through Programming by Demonstration (PbD). It is necessary to compare this mainstream paradigm with our proposed framework. It

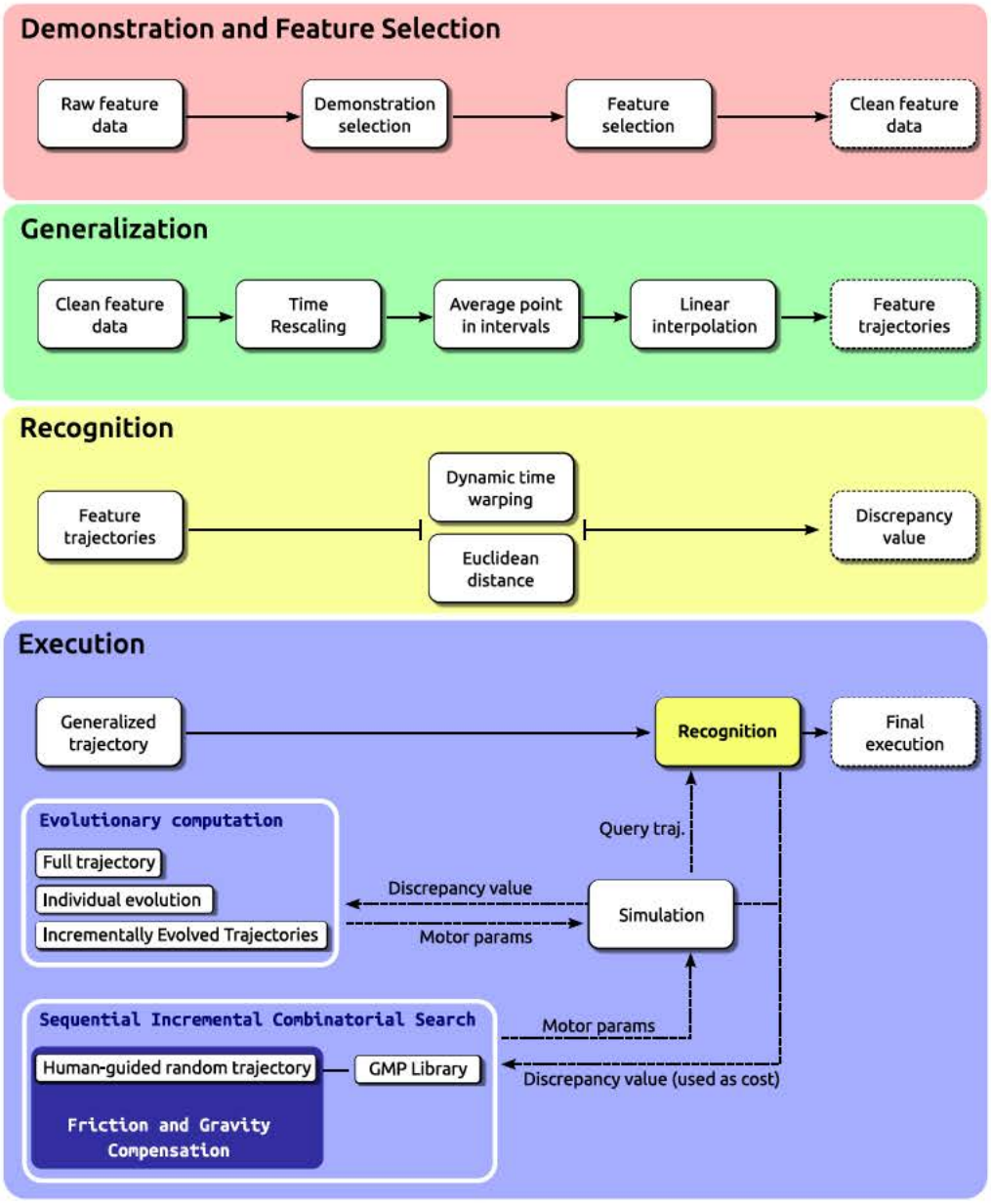

Figure 3.1: *Block diagram of the thesis. The block order defines the temporal order of the process. First, a demonstration and feature selection is performed. Then comes the generalization. The next step is to be able to perform recognition, or comparison, among feature trajectories. Finally, and using the information from the previous blocks, action reproduction can be completed. There are two main alternatives for the joint parameters generation: to use evolutionary computation or to combine guided motor primitives. This last option uses the friction and gravity compensation controllers developed.*

is important to notice that both paradigms are not exclusive, but complementary. The main differences between the CGDA and PbD paradigms can be found in Table 3.1.

Table 3.1: *Main differences between PbD and CGDA paradigms.*

	PbD	*CGDA*
Objective of imitation	Spatial trajectories	Object feature states
Features tracked	Demonstrator's joint positions/velocities	Object's shape, area, color, coordinates, etc.
Strengths	Perfect kinematic imitation	Effects encoding
Weaknesses	Undefined goal to achieve	Undefined way to achieve the goal

The CGDA framework can be used for generalizing, recognizing and executing goal-directed actions by analyzing their effects on objects, or more generally on the environment. A continuous analysis generates a trajectory in an n-dimensional feature space, where n equals the number of scalar values that correspond to the tracked object features. Examples of tracked features are color, area, weight, spatial positions, etc. The trajectories are discrete due to sampling rates. Thus, action repetitions lead to a point cloud through time. This point cloud is the subject of analysis of the developed techniques.

An action encoded as CGDA allows a seamless transfer of actions between dissimilar robots. As the encoding does not depend on the robot configuration and only depends on the action effects on the environment, the encoded action can be executed by any robot, provided a way to execute CGDA actions.

The rest of the chapter is dedicated to provide a high-level understanding of the inner workings of CGDA and its complementary extensions. This section, and the remaining ones of this chapter, offer a brief overview of the whole framework. Motivations, justification of choices made, in-depth analysis of algorithms and diagrams are not provided in these sections, and we recommend the reader to jump to each specific chapter for clarifications in the explanations (links are provided in the text).

3.2 Selecting Relevant Demonstrations and Features

As stated previously, the set of demonstrations provided by the user, may include inadequate demonstrations and unnecessary features. A logical step is to include a block previous to the generalization block to filter them.

From the raw data, the logical temporal order of screening is to first discard unwanted demonstrations, and then select the relevant features. Let us explain why. If the feature selection were performed in the first place, **features from erroneous demonstrations would influence the results**, potentially discarding relevant features and conserving irrelevant ones. See 4.2 and 4.3.

One may argue that by first screening the demonstrations, the opposite argument could be presented. It is undeniable that **irrelevant features influence the demonstration selection**. However, from our experience, if a demonstration is performed incorrectly, it influences many features in a noticeable way. This empirical fact facilitates the detection of the most different demonstrations of the set.

3.3 Basic CGDA Framework

Any complete goal-directed framework has to include, at least:

- A generalization module, to create a representative model of the task.
- A recognition module, to be able to measure the similarity of two tasks.
- An execution module, to reproduce an action.

The basic framework flows as follows: **Human demonstrations are represented by a sequence of discrete points in the feature space**. The set of demonstrated action repetitions leads to a point cloud in the feature space. A representative feature trajectory is extracted from the cloud. **This feature trajectory represents changes produced in the object features when an action is performed on it**. Generalization is composed by the following three steps.

1. Time Rescaling: Before inserting an action repetition in the point cloud, each repetition must be **normalized** in time in the range $[0, 1]$. All normalized feature trajectories are introduced in the same object feature space, forming a point cloud. See 5.1.1.

2. Average in Temporal Intervals: To model the point cloud, we split it in temporal intervals (e.g. one interval per second). The representative point of each interval is the **interval average point**. When applied to all the intervals, the result is a vector of interval average points. See 5.1.2.

3. Interpolation: Once we have each interval average point, we have to connect them to create a **generalized feature trajectory** of the action. As an interpolator, we use a linear Radial Basis Function (RBF) which returns a generalized feature trajectory. See 5.1.3.

With these steps we obtain a generalized feature trajectory from a set of repetitions. This generalized trajectory represents the **changes produced in the object**. In this high dimensional trajectory, each dimension is an object feature.

Recognition is performed by comparing feature trajectories. Two distance measures have been tested Dynamic Time Warping or Euclidean distance. Independently of the measure chosen, it returns a **cost of adjusting the trajectories**. This cost can be used to decide if two actions are similar or not. See 5.2.

Execution is achieved by relying on evolutionary computation to generate the motor trajectories. Several evolutionary strategies have been developed. See 6.1. Once we have a good model of the task, and we are able perform a basic execution, research focus can be directed to improving execution, leveraging the physical capabilities of the robotic platform. Reaching this objective may increase the naturalness of movements.

3.4 Generating a Library of Motor Primitives

One way to improve the reproduction of actions is to increase the naturalness of robot's movements. This is possible, among other alternatives, if the robot makes use of a library of motor primitives. From this library, a selection of primitives must be made, and an ordering of primitives created in order to achieve the desired environment effects.

In this thesis, we have approach primitives this way: **The human physically interacts with the robot** by guiding the task-relevant robot limbs in a random way. The analysis of the movement provides a database from which **small chunks of movements are extracted**, selected and organized to form a library of motor primitives. It is the

combination of these primitives that generate the correct movements to complete the tasks.

To select and order the primitives, an algorithm has been developed. This algorithm is a custom **tree search** algorithm. Topologically, **each node represents a primitive**, and **each edge is the cost of executing the primitives** from the initial node to the current one. This cost is evaluated in a simulator. See 6.2.

To generate the primitives in this way, the human has to physically move the robot. Unfortunately, most humanoid robots are heavy and present high friction, due to the reduction gears and the mechanical elements. Additional controllers with friction and gravity compensation terms are needed to provide a more natural, and more comfortable, fashion to generate motor primitives.

3.5 Compensating Friction and Gravity

Controllers have been developed that alleviate the burden of motor primitive creation. It is important to notice that **these controllers are partial compensators that use simple friction and gravity models**. This thesis considers the **use-case of Programming by Demonstration** and the controllers are adapted to this scenario. They are adequate controllers for **low joints velocities**.

Two controllers have been designed. The Low-Friction Zero-Gravity controller allows a **guidance of the robot without effort**, allowing small friction forces to reduce the free robot motion. It can serve to aid users providing kinesthetic demonstrations while programming by demonstration. In the present, kinesthetic demonstrations are usually aided by pure gravity compensators, and users must deal with friction.

A Zero-Friction Zero-Gravity controller results in **pseudo-free movements**, as if the robot were moving without friction or gravity influence. Ideally, only inertia drives the movements when zeroing the forces of friction and gravity. In reality, **this controller is able to maintain this behavior up to the joint limit**. It is probably not modeling the friction well enough to maintain the movement indefinitely. Coriolis and centrifugal forces are depreciated.

The review presented in this chapter can help the reader in understanding the framework from a high-level perspective. It may also serve as a summary to be consulted if the explanations saturate the reader comprehension in the following chapters.

Chapter 4

Demonstration and Feature Selection

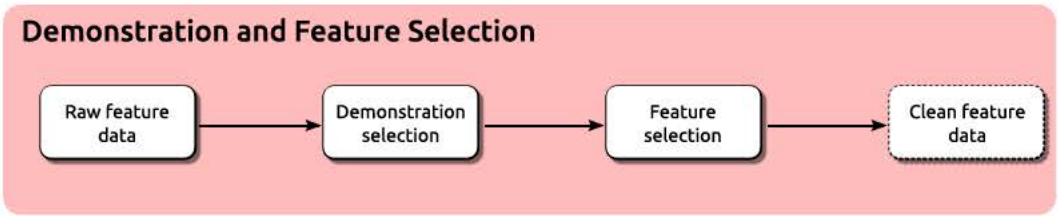

Figure 4.1: *Demonstration and Feature Selection block.*

Robot learning frameworks provide a way for non-expert users to teach robots new actions.

Proposition 1 *All Robot learning frameworks share two key factors for which they can be identified: the* ***user demonstrations*** *are the main input to the framework, and they internally generate a* ***generalized model*** *of the task.*

A demonstration selection process to filter bogus performances or incorrectly sensed user demonstrations can be incorporated as an initial step in any of these frameworks. Regarding the generalized model of the task, we can classify the main approaches to generalizing tasks in robot imitation:

- Programming by Demonstration (Calinon, 2009) conventionally encodes a generalized action as a probabilistic model in the robot joint or operational space.
- Dynamic Movement Primitives (Schaal, 2006) encodes a generalized action as a control policy, generally in the operational space.

- Continuous Goal-Directed Actions (Morante, Victores, Jardón, & Balaguer, 2014) encodes a generalized action as a feature trajectory preserving all the scalars that can be extracted from the sensor data at each instant.

Proposition 2 *Attending to the space where the generalized model is stored, a handcrafted feature selection process is implicitly performed when defining the structure of the generalized model of the task.*

While reducing an action to the joint or operational space is a clear over simplification for many use cases (e.g. filling a glass depends on the layout of the environment), preserving all the scalar features that can be extracted from the sensor data can lead to not knowing which feature is relevant for the task. For instance, a person fills a glass with water by pouring it from a bottle. Which are the features that are relevant for the task? The area of the glass that is perceived with a slightly different color due to the new refraction index? The absolute or relative position and orientation of the bottle? Reproducing the sound of a motorcycle that was passing by during one of the demonstrations?

For this problem, this thesis presents a solution: **choose as relevant signals (features or demonstrations) those consistent among task repetitions**. This dilemma, deciding if a consistent signal is relevant or not, dates back to Newton's Philosophiae Naturalis Principia Mathematica (Newton, Bernoulli, MacLaurin, & Euler, 1833). In his book, in addition to introducing the law of universal gravitation, Newton stated four rules of reasoning in philosophy, which can be called rules of induction because of their content. Newton offered a methodology for dealing with unknown phenomena and building explanations for them:

1. We are to admit no more causes of natural things than such as are both true and sufficient to explain their appearances.
2. Therefore to the same natural effects we must, as far as possible, assign the same causes.
3. **The qualities of bodies, which admit neither intensification nor remission of degrees, and which are found to belong to all bodies within the reach of our experiments, are to be esteemed the universal qualities of all bodies whatsoever.**

4. In experimental philosophy we are to look upon propositions inferred by general induction from phenomena as accurately or very nearly true, not withstanding any contrary hypothesis that may be imagined, till such time as other phenomena occur, by which they may either be made more accurate, or liable to exceptions.

Specially interesting for our problem is the rule number three. Rephrasing it in modern language, it says:

> Whatever holds true for every case we have seen, holds true for all the other cases.

This rule is at the heart of many (if not all) machine learning algorithms, and it is also of our selection process. In this thesis, a machine learning process for variable selection is presented. The process is called Dissimilarity Mapping Filtering (DMF). DMF is flexible in its design, **allowing different algorithms in each of its blocks**. DMF is applied for demonstration selection and for feature selection.

4.1 Dissimilarity Mapping Filtering

The Dissimilarity Mapping Filtering process (DMF), applied as a demonstration selector and as a feature selector (Morante, Victores, & Balaguer, 2015a), is introduced in this section. An additional preprocessing step is incorporated in the descriptions prior to the dissimilarity, mapping, and filtering steps. A visual example of a generic DMF use is shown in Figure 4.2. In this picture, the term *signal* represents either demonstrations or features.

Despite the DMF process is generic, non-specific for robot learning, and can be applied to other problems, we aim at solving a problem in a target scenario. We work in a robotics learning scenario, using a humanoid robot equipped with simple 3D vision, learning a block-moving task (see Figure 4.3).

4.2 Demonstration Selector

Using DMF as a demonstration selector, **the goal of the process is to automatically discard incorrectly sensed or performed demonstrations**. Assuming each fea-

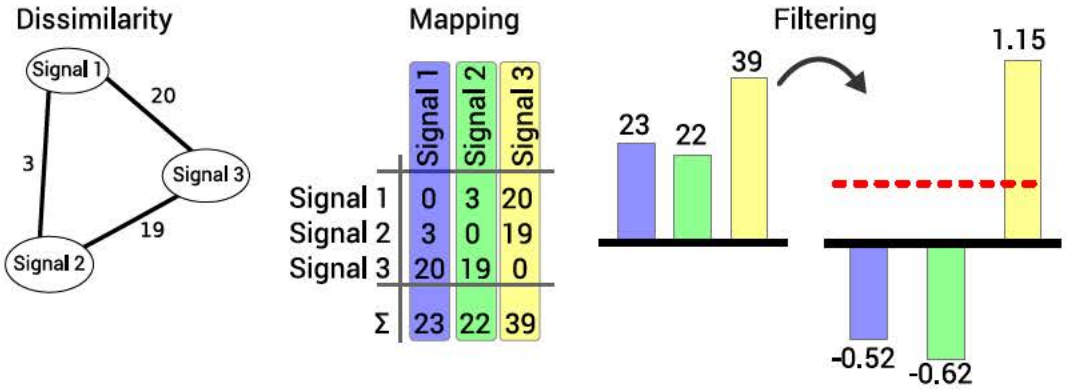

Figure 4.2: *Imagine we want to screen the most irrelevant of the three signals. First, we extract a measure of dissimilarities among them. These dissimilarities can be represented as nodes' distances in a graph. Then, we map these distances from a matrix to a single dimension e.g. by summing the columns. In the last step, we can represent values in a bar plot. To further increase their differences, a transformation, e.g. z-score, is applied. This transformation allows to better distinguish the irrelevant signal, which can now be filtered with a threshold (represented by a dotted red line).*

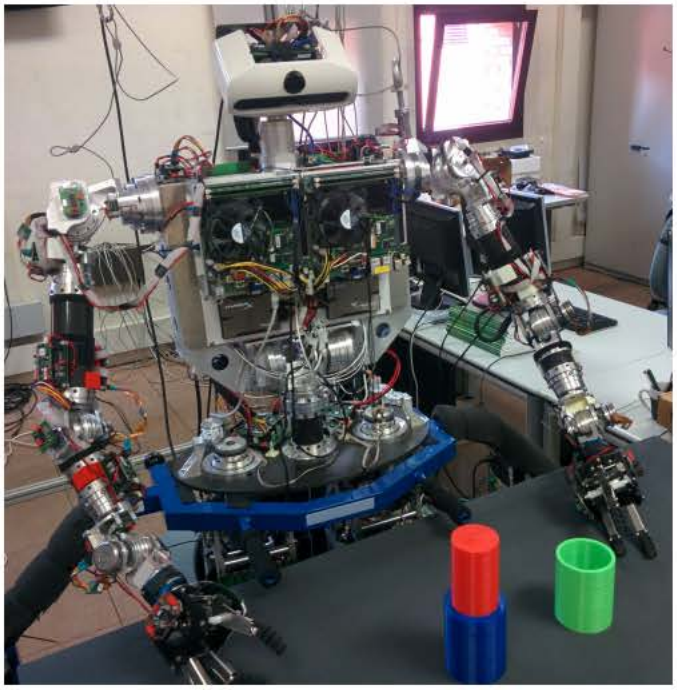

Figure 4.3: *Humanoid robotics learning scenario: block-moving task.*

ture is a time-varying scalar value, the set of user demonstrations that is fed to DMF can be treated as a set of multidimensional signals. Examples of features are: joint an-

gles, centroid position coordinate, or more arbitrary features supported by the CGDA framework, such as percentage of area of a certain color (Morante, Victores, Jardón, & Balaguer, 2014).

4.2.1 Preprocessing

First, the input data is normalized. While **normalization may not be required** if all the features are presented in similar ranges, mixing joint angles in conjunction with operational space measurements would require weighting the variables to be comparable. Four different types of normalization are now presented. Each choice can affect the final demonstration selection results. Depending on the field of study where DMF is applied, it may be convenient to use one of these normalizations, or none.

- MinMax: Each feature, for each demonstration is normalized within the limits of the feature:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{4.1}$$

- Standardized: Each feature is standardized:

$$X_{norm} = \frac{X - X_{mean}}{X_{stddev}} \tag{4.2}$$

- Whole Experiment Normalization: Each feature is scaled by the maximum value for this feature, among all the demonstrations:

$$X_{norm} = \frac{X}{X_{maxExperiment}} \tag{4.3}$$

- Physical Limits: Each feature is normalized within the physical limits of the sensor that provides the information for this feature:

$$X_{norm} = \frac{X - X_{PhysicalMin}}{X_{PhysicalMax} - X_{PhysicalMin}} \tag{4.4}$$

The next step after the normalization is to reduce the multidimensional complexity. Two different approaches are considered for reducing each multidimensional signal into a one dimensional signal. Dimensionality reduction is a well studied field in machine learning (Alpaydin, 2014). Methods like Principal Components Analysis (PCA) or Multidimensional Scaling (MDS) have become classic techniques when facing a high dimensional problem. However, additional restrictions led us to look for an alternative method:

1. In this step, only demonstrations are being screened, so **no feature can be eliminated in the process**. The comparison would not be fair, if different demonstrations were compared using different features. Therefore, all of the features must be taken into account when discarding demonstrations. **This fact prevents us from using any method of subset selection**, also called shrinkage methods, which would keep only a subset of features to reduce dimensionality (Friedman, Hastie, & Tibshirani, 2001).

2. An **equivalent reduced space needs to be extracted from each demonstration**. This fact **prevents us from using PCA, MDS, and derivatives**. If used, these methods would create a different reduced dimensional space for each demonstration, complicating their comparison.

For these reasons, alternative methods have been developed. The reduced dimensional space is easier to analyze if it is 1-dimensional. The first approach considered is to sum all the time steps of the same demonstration into a single signal, by summing all the discrete points of each feature signal, see Figure 4.4. Performing the same operation on all the demonstrations, we obtain a single signal per demonstration of size $1 \times M$, where M is the total number of features. This method is called *summing rows* throughout the thesis.

The second approach reduces the demonstration into a single signal by summing all the features for each point, obtaining a signal of size $N \times 1$ where N is the total number of points of the demonstrations. This method is called *summing columns*.

Whichever the method selected, after this step, each user demonstration has been converted, from a multidimensional signal, into a **one dimensional signal** that can be used in the next step.

4.2.2 Dissimilarity

Dissimilarity is the step of **obtaining a measure of how different two demonstrations are**. This measure may be an absolute measurement of dissimilarity, or a relative one between two signals. DMF is a flexible process where the designer can select a different algorithm for each of the steps.

Dynamic Time Warping (DTW) has been selected to obtain a distance matrix (Albrecht, 2009), but other measurements can be used (e.g. Euclidean distance, uniform scaling,

Summing rows (1xM)

t	f_1	f_2	f_3
0	56	65	9
1	55	65	3
2	57	62	2
3	53	60	0
4	54	59	1
5	52	59	1
Σ	327	370	16

Summing columns (Nx1)

t	f_1	f_2	f_3	Σ
0	56	65	9	130
1	55	65	3	123
2	57	62	2	121
3	53	60	0	113
4	54	59	1	114
5	52	59	1	112

Figure 4.4: *Different preprocessing methods for reducing multidimensional information.*

etc.). DTW is used for measuring the similarity between two temporal signals which may vary in duration or speed. The sequences are "warped" (hence the name) non-linearly in the time dimension to determine a measure of their similarity independent of certain non-linear variations in the time dimension. One advantage of DTW is that it accepts demonstrations of different durations as input. Moreover, **in our case, synchronization issues among features of a demonstration are avoided when columns are summed in the preprocessing step**, as they collapse in each time step into a single value.

Let us describe how DTW works. Let $X = \{x_1, .., x_p\}$ and $Y = \{y_1, .., y_q\}$ be two generic 1-dimensional signals. To compare two elements within the signals, a local cost measure ($dist(x, y)$, e.g. Euclidean distance) is needed. A lower cost represents a bigger similarity of the sequences. Evaluating all pairs of points between the signals, a cost matrix CM is obtained:

$$CM = \begin{pmatrix} dist(x_0, y_0) & \cdots & dist(x_p, y_0) \\ \vdots & \vdots & \vdots \\ dist(x_0, y_q) & \cdots & dist(x_p, y_q) \end{pmatrix} \tag{4.5}$$

The goal now is to find the lowest cost alignment path between the signals. This path is usually calculated in an accumulated cost matrix derived from CM, where each cell represents the distance of the correspondent pair $dist(x, y)$ plus the cost to reach this cell (see Figure 4.5).

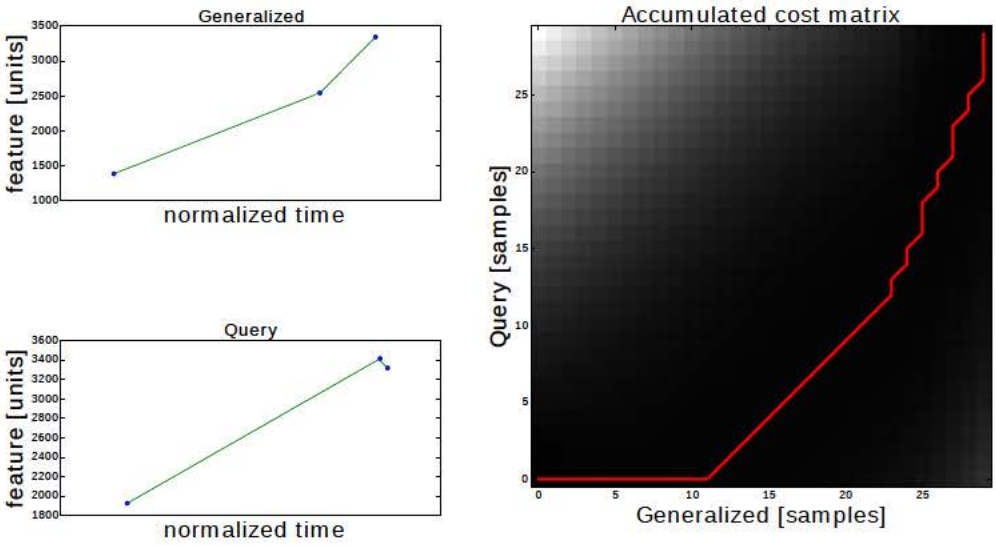

Figure 4.5: *Example of accumulated cost matrix for two sequences. On the left side, the interpolated sequences are shown. The right side depicts the Dynamic Time Warping computation. White cells represent high cost, while dark cells are low cost ones. The red line is the lowest cost path.*

In this accumulated matrix, the cost of alignment $C(X,Y)$ of the optimal path is:

$$C(X,Y) = \frac{C(x_p, y_q)}{P+Q} \tag{4.6}$$

DTW is applied for the comparison of each pair of demonstrations, obtaining one cost C in each comparison. With these costs, the distance matrix DM is created for all demonstrations d:

$$DM = \begin{pmatrix} C(d_0,d_0) & \cdots & C(d_R,d_0) \\ \vdots & \vdots & \vdots \\ C(d_0,d_R) & \cdots & C(d_R,d_R) \end{pmatrix} \tag{4.7}$$

Where R is the total number of demonstrations of the experiment. This symmetrical matrix DM **represents the dissimilarity among demonstrations** and is the output of this block.

4.2.3 Mapping

The main idea in this step is to **reduce the information** provided by the previous step, which is a matrix, into a **single scalar value per demonstration**. This reduction can be considered a mapping into a lower dimensional space.

Let us remind what this matrix is. Each cell i, j in the matrix represents the dissimilarity between the demonstration i and the demonstration j. When $i = j$, this dissimilarity will be, obviously, zero.

The aim is to **compress into a single value, how dissimilar a demonstration is with respect to the rest of the demonstrations** of the set. The matrix DM provides this information by collapsing (summing) its rows or columns (being both results equal as the matrix is symmetrical). In an alternative interpretation, if the signals are imagined as nodes in a graph, the dissimilarity between two signals is the "distance" that separates them, as shown in Fig. 4.2. By summing all the distances from a single node, to all the other nodes, we are capturing how far this signal is from the rest of signals.

We have chosen to map the distance matrix DM into a single dimension by summing the values in the columns, but other algorithms can be used, e.g. multidimensional scaling, ISOMAP, etc. With our method, a value V is obtained for each demonstration i:

$$V_i = \sum_{j=0}^{R} C(d_i, d_j) \tag{4.8}$$

Each value V represents a **measurement of the dissimilarity of a demonstration with respect to all the others**, and R is the total number of demonstrations. The set of measurements of dissimilarity is the returned data of this block.

4.2.4 Filtering

In this last step, the aim is to screen the demonstrations, to **discard the outlying ones**. If previous algorithms have been correctly designed and selected, and have not distorted the information contained in the demonstrations, there should be possible to isolate the most different ones of the set.

We have chosen to filter them by using the z-score (called standard score) because it standardizes the results, but other algorithms can be used: DBSCAN, t-test, etc. In

fact, any algorithms able to isolate outliers in one dimension can be applied. In case the standard score is selected, each value V is converted into a z-score Z:

$$Z = \frac{V - V_{mean}}{V_{stddev}} \tag{4.9}$$

Finally, the demonstrations with a z-score Z higher than a threshold α are discarded. In common statistical outliers detection, the positive threshold has a symmetrical equivalent on the negative side, to discard the values lower than the negative threshold. In the filter presented in this section, there is no need of a symmetrical threshold. The reason is that values to be standardized represent dissimilarities, and we are only looking for the biggest dissimilarities of the set. This threshold α is the only tunable parameter of the whole DMF process, if the designer chooses the same algorithms as in this thesis.

The use of z-score as filtering method implies accepting an assumption in the data: there are enough relevant signals (in this case, demonstrations) in the set to compensate the effect of irrelevant ones. The reason behind this assumption is that in the process of subtracting the mean to all the values, it may occur that an abundance of irrelevant demonstrations rises up the mean value, mitigating the effect of the transformation, and allowing irrelevant signals to be considered as relevant.

This assumption should not be a problem for demonstration selection, because if a user is demonstrating a task to a robot, and most of the demonstrations are incorrect, maybe this user is not the best teacher for the robot. A possible approach for solving this problem would be to apply iteratively the filtering algorithm several times (two times should be enough for most cases). It will reduce the number of irrelevant signals, but may potentially discard relevant ones if it is applied many times.

This preprocessing, dissimilarity, mapping and filtering process, discards the incorrect demonstrations. Once the set of demonstration is curated, it is possible to filter irrelevant features. In the next section, how to discard irrelevant features using DMF is explained.

4.3 Feature Selector

The goal of this section is to **discard those features not relevant for a specific task**. We start again from the raw data, but **using only the correct demonstrations** selected in the previous step. This machine learning pipeline is automatic, as each step can feed the

next one without human intervention or curation. What we propose in this section is to use the same DMF process as in the previous section, but applied to feature selection.

4.3.1 Preprocessing

As in the previous case, there may be the need for normalization. The possibilities are the same as those in the previous case. However, there is no need to reduce the multidimensional complexity in this case. The reason is that each feature, for each demonstration, is already 1-dimensional. No further dimensionality reduction is necessary.

4.3.2 Dissimilarity

Again, DTW has been chosen for measuring dissimilarity. In this case, the aim is to obtain a cost value for each feature. Let us extend on this. To decide which features are the ones to be discarded, we assume there is consistency among all the demonstrations, for the relevant features. In the same way, we assume that irrelevant features will differ among demonstrations. Each feature is mathematically a time-varying scalar value.

For instance, imagine the analysis were focused to analyze an object's *area* and its *position* for an action repeated twelve times.

1. First, the twelve 1-dimensional *area* trajectories would be placed on the same feature space for analysis.

2. Second, the twelve 1-dimensional *position* trajectories would be placed on another feature space for analysis.

3. From each feature space, we calculate the dissimilarity among all demonstrations. We obtain one cost C every time two trajectories are compared.

4. With these costs, we create the distance matrix DM for all demonstrations d for each feature (similar to Equation (4.7)). One distance matrix is obtained per feature space.

A graphical explanation of the dissimilarity analysis for a simpler case can be found in Figure 4.6.

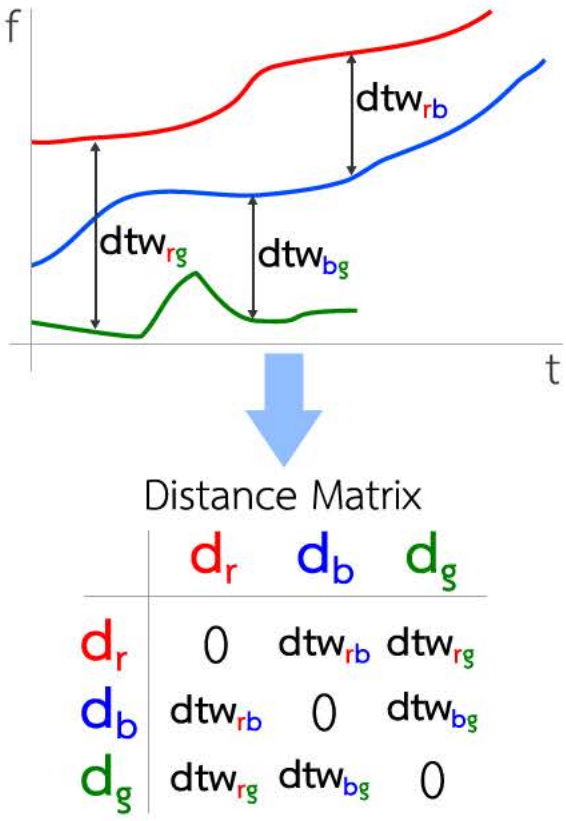

Figure 4.6: *For each feature space, a distance matrix is calculated. Each color line (red, blue and green) is the same feature but extracted from a different demonstration. When represented in the same space, these features trajectories will show some kind of similarity if they are relevant, or will be different if they are not.*

4.3.3 Mapping

The mapping in this case aims to obtain a single value that represents a global measure of how different all the trajectories in the same feature space are. This global measure is obtained as the sum of all the elements of the distance matrix, resulting in a single value V_{total} that represents a measurement of the dissimilarity of the demonstrations for a given feature:

$$V_{total} = \sum_{i=0}^{R} \sum_{j=0}^{R} C(d_i, d_j) \tag{4.10}$$

At this point, a set of values has been obtained, one per feature space. As in the previous case, the next step is to discard the outlier ones.

4.3.4 Filtering

The filtering process is again performed using z-scores. Each value V_{total} is converted to a z-score Z, and the algorithm discards the features with a z-score Z higher than an α threshold.

As in the previous case, the use of z-score as filtering method implies there are enough relevant signals (in this case, features) to compensate the effect of irrelevant ones. This assumption may potentially be a problem for feature selection. If a system is endowed with very general task-independent features, it is reasonable to think that most of them will not be relevant for a specific task. The solution offered in this case is again, to apply iteratively the filtering algorithm several times. Another obvious solution for these cases is to use another algorithm in the filtering block.

After this whole process of demonstration and feature selection, a clean and curated dataset has been obtained, and the CGDA analysis can be carried out.

Chapter 5

Generalization and Recognition

This chapter presents the core of CGDA. The CGDA framework develops around the idea of analyzing the effects of an action on the environment. Every robot learning framework assumes a model of the demonstrated task. To create this model, a generalization of the whole set of demonstration has to be done. Even with a generalized model of the task, if the framework aspires to be useful, it has to include a method for model comparison for recognition. This is essential, not only to compare two actions, but even for evaluating the correctness of the action reproduction.

5.1 Generalization

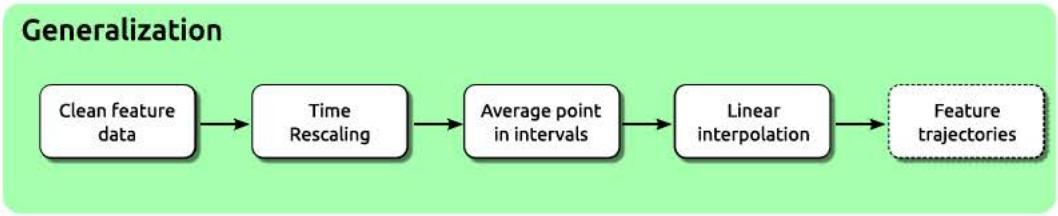

Figure 5.1: *Generalization block.*

For generalization purposes, we need to **extract a representative n-dimensional trajectory of the point cloud**, with n being the number of features to generalize. The point cloud is composed by the discretized trajectories from the whole set of repetitions. This process is composed by the following three steps.

5.1.1 Time Rescaling

Before inserting a single action repetition in the point cloud, it is **normalized in time** (range $[0, 1]$). With this time rescaling, every action demonstration gets **bounded by the same temporal limits**, making the algorithm independent of the repetitions' speed. While this decision simplifies the posterior analysis, it makes a potential execution unaware of the task's speed. A **preprogrammed execution speed** should be provided, **or speed itself should be considered as an input feature** to the system. All normalized trajectories are introduced in a feature space, forming a point cloud.

5.1.2 Average in Temporal Intervals

To model the point cloud, we split it in temporal intervals, fixing **one interval per second**. The number of seconds is computed from the average duration of the original repetitions. Each interval contains points of all repetitions, in the same percentage of execution. As the repetitions are normalized in time, each interval represents a percentage of action execution, and **the number of intervals allows preserving a notion of the action duration**. The representative point p of each interval is extracted as:

$$p = \frac{1}{p_{int}} \sum_{i=0}^{p_{int}} X_i \tag{5.1}$$

Where p_{int} is the number of points in the interval and X_i represent the vector of features for a point. In other words, this representative point of each interval is extracted as the average for each dimension of all points of the interval, as seen in Figure 5.2. The result is a vector of average features. These temporal **averaged points are the ones used in the posterior interpolation**.

5.1.3 Radial Basis Function Interpolation

Once the representative points of the point cloud are obtained, they have to be joined to create a generalized action model, i.e. an object feature trajectory we can consider as a generalization. In a robot joint space, an interpolation could create a jerky joint trajectory, so literature, e.g. (Calinon et al., 2010), commonly uses regressors such as Gaussian Mixture Regression. However, working in the object feature space, we use an **interpolator to assure the trajectories pass through the target points** (which are the

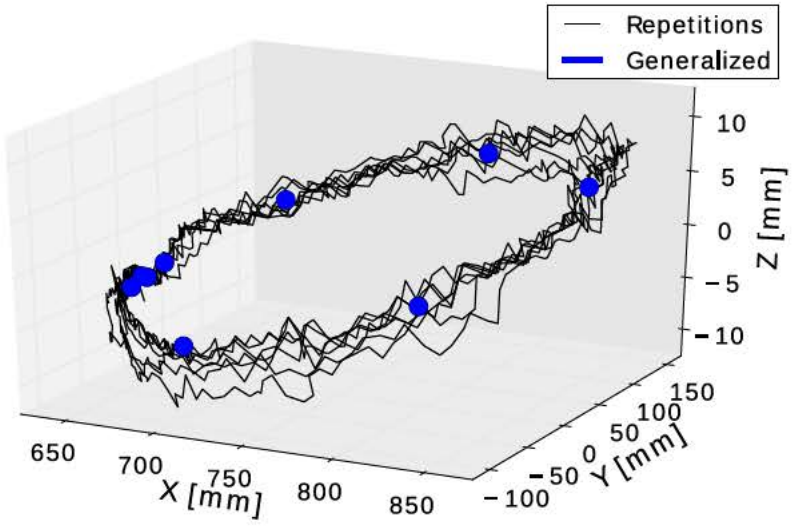

Figure 5.2: *Plot representing a three feature trajectory. Black lines are training action repetitions. The blue dots are the averaged points.*

states of the object in an instant). We use a Radial Basis Function (RBF), which is an interpolation technique based on measuring the influence of every known point over the queried point (Press, 2007). The RBF interpolation $f(x)$, which will become the final generalized trajectory, is mathematically expressed as a sum of radial basis functions:

$$f(x) = \sum_{i=1}^{N} w_i \, \phi(\|x - x_i\|) \tag{5.2}$$

Where N is the number of radial basis functions, equal to the number of intervals, and x_i represents the coordinates of each interval's known point. The radial basis function is denoted as ϕ, where the input parameter is the distance between the known point x_i and the queried point x, measured with L^2 norm. The coefficient w_i is the weight of a specific known point over the queried point x, and it is the value to be solved. As the interpolation is known at known points, the weight problem is solved as a set of N linear equation with N unknowns:

$$
\begin{aligned}
f(x_1) &= \sum_{i=1}^{N} w_i \, \phi(\|x_1 - x_i\|) \\
&\vdots \\
f(x_N) &= \sum_{i=1}^{N} w_i \, \phi(\|x_N - x_i\|)
\end{aligned}
\tag{5.3}
$$

From the available radial basis functions, we have selected the linear one (because we do not care much about smoothness on the feature space):

$$\phi(r) = r \tag{5.4}$$

Where $r = \|x - x_i\|$. Once the interpolated function is returned, we consider this output as the generalized function of an action. Its physical meaning is how the state of the object, regarding its features, is changing across the action performance. A final generalization example, once this process is completed, can be seen in Figure 5.3.

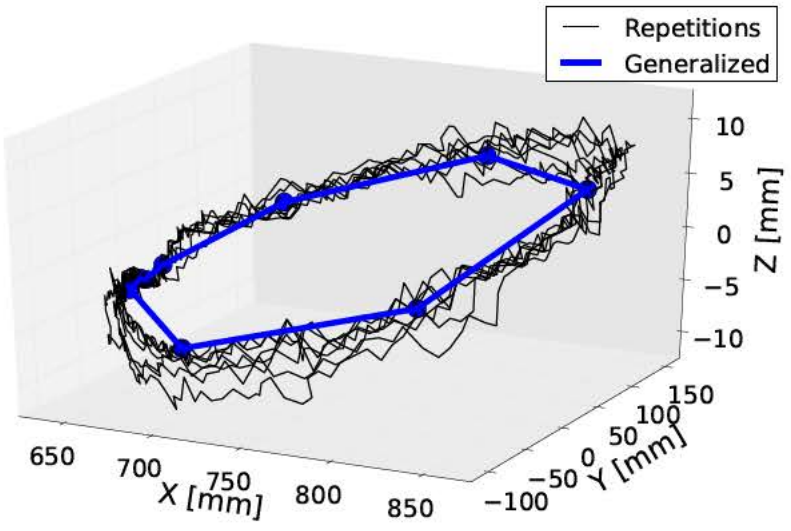

Figure 5.3: *Plot representing a three feature trajectory. Black lines are training action repetitions. The blue line is the generalization of all the repetitions.*

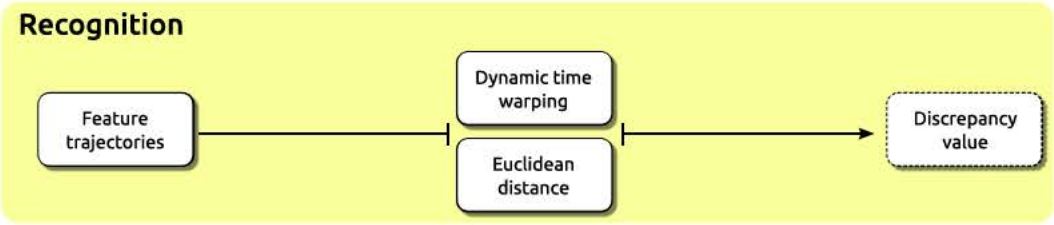

Figure 5.4: *Recognition block.*

5.2 Recognition

The generalized trajectories can be used for recognition. Recognition can be performed by **comparing a query trajectory** (a demonstrated action) **with the previously computed generalized trajectories** (action models), returning the one with the highest similarity. This application can be useful for composing tasks as sequences of smaller and simpler actions. To recognize actions, it is assumed we have a database of generalized actions, and an unknown query action that we want to associate to the most similar of the database. We assume recognition is a comparison of a query with our generalized trajectories, returning the one with the highest similarity.

Looking for flexibility, two different methods to perform recognition have been implemented. One of them is more flexible, but can lead to inaccuracies (Dynamic Time Warping), and the other is more rigorous but is affected by variations between the query and the generalization (Euclidean Distance). The initial processing of the query action is the same as explained in the generalization section (time rescaling, average in temporal intervals and RBF).

5.2.1 Dynamic Time Warping

As the query and the generalized actions are normalized in time, we can take t values along time for each action to compare them. The technique used for comparison is Dynamic Time Warping (DTW), which is an algorithm usually used to optimally align two temporal sequences (Albrecht, 2009). DTW was already explained in detail in Section 4.2.2.

In our case, we use the normalized cost of the optimal path (as in Equation (4.6)) as a measure of discrepancy between dimensions. As DTW is computed between two signals for one dimension only, we consider the **total cost of alignment between two**

n-dimensional trajectories as the sum of the costs of the optimal paths of each dimension, obtaining a single score D:

$$D = \sum_{i=1}^{n} C_{P_{norm}}(X_i, Y_i) \tag{5.5}$$

This score is used as the measure of discrepancy between two trajectories in the n-dimensional space. In recognition, the trajectory with the smallest score is the one we consider the match.

5.2.2 Euclidean Distance

Another option to perform recognition, or in general, to measure the error between two trajectories, is to use the Euclidean distance. The query trajectory is normalized in time, in the same way it was for the generalized ones. This step allows us to take t values along time for each action (the query and the generalized) to compare them. The use of DTW is motivated to allow a non-rigid measure of similarity between trajectories, as DTW is able to 'tighten' and 'widen' until both are best aligned. Unfortunately, DTW allows time displacements that can affect the order of execution of actions, resulting in task failure or performance decay. Therefore, Euclidean distance can be considered an improvement with respect to DTW as metric.

Our aim when using Euclidean distance is to obtain a **metric of discrepancy between two time-dependent sequences of points**, namely $X = \{x_1, .., x_N\}$ and $Y = \{y_1, .., y_N\}$. We compute the total discrepancy $J(X, Y)$ between two n-dimensional trajectories as:

$$J(X, Y) = \|X - Y\| \tag{5.6}$$

The generalized trajectory with the smallest $J(X, Y)$ is the one we consider as the most similar. Let us recall that all the trajectories involved in this process of recognition are feature trajectories and not joint space trajectories. This implies that we are recognizing actions by what is happening to the object, and not comparing human movements.

The differences in comparison of signals between Dynamic Time Warping and Euclidean distance can be seen in Figure 5.5.

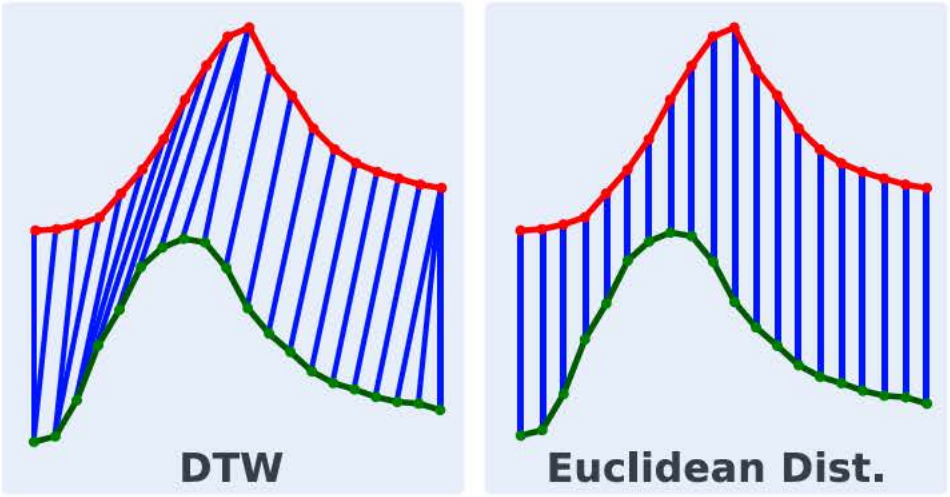

Figure 5.5: *Two alternatives are presented to be used as distance measures.*

Independently of the distance used, in this section we have obtained a method for evaluating the correctness of an execution. With this tool, methods for execution can be designed.

Chapter 6

Execution

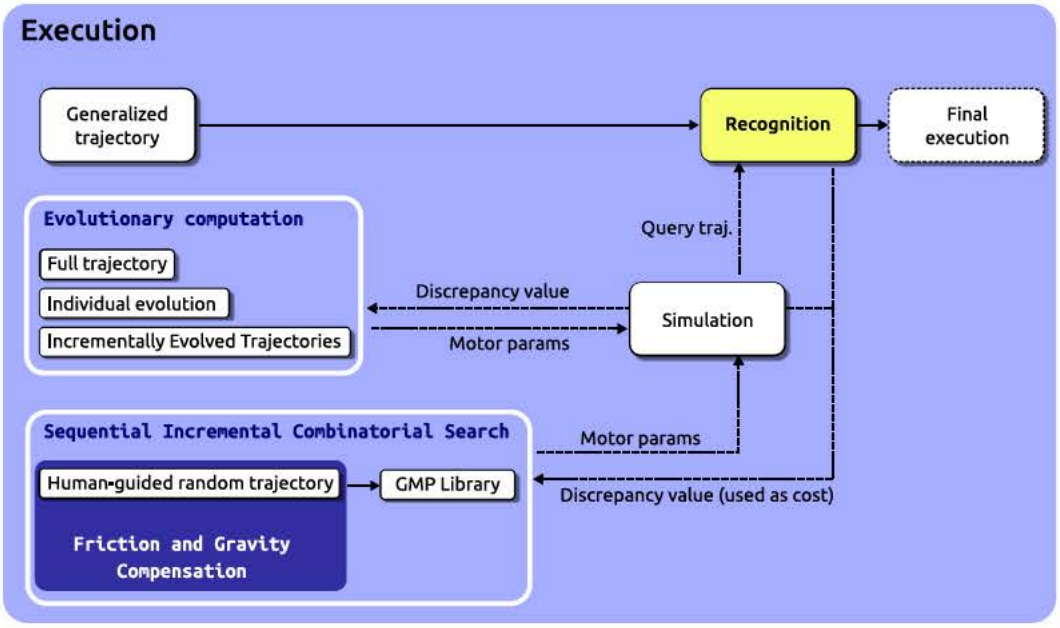

Figure 6.1: *Execution block.*

Probably the most useful application for an imitation system is the robot reproduction of the learned (generalized) action. Tasks encoded as CGDA require a particular technique for performing the feature trajectory. As CGDA does not encode joint motor parameters, only object feature changes, the robot must be provided with a method for assuring the task accomplishment. **Object spatial trajectories are computable with inverse kinematics** (IK), but if we want to **extend this framework to other features** like

color or area, and we also want them to be robot-configuration independent, we need to choose another method.

Several algorithms are tested in this thesis. We can group them into two options for robot reproduction:

- One tested algorithm is **Evolutionary Computation** (EC) applied on a humanoid robot in a simulated environment. Several evolutionary strategies have been developed for obtaining motor trajectories.

- Another tested approach is the creation and ordering of motor primitives through a tree search algorithm (**Sequential Incremental Combinatorial Search**).

6.1 Simulation using Evolutionary Strategies

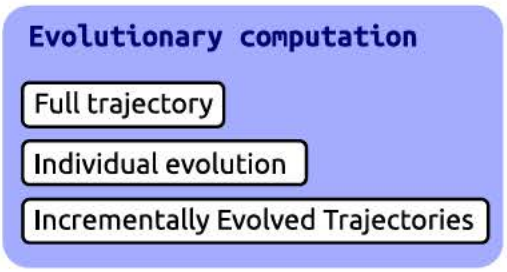

Figure 6.2: *Evolutionary block.*

In this section, **evolutionary computation is applied to generate motor joint parameters**. Evolution guides the motor joints to places where the evaluation of the position obtains a high reward (fitness). In the target feature trajectory is discretized and the points are considered a series of ordered targets, evolutionary computation can generate joint values adequate for the task. Three evolutionary strategies are presented:

- Full Trajectory Evolution strategy: The **whole trajectory is evolved** at the same time.

- Individual Evolution: **Each point is evolved independently**, but temporal order is not considered.

- Incrementally Evolved Trajectories: This strategy has been specially crafted for features which have a **temporal dependency**, which is the case of many non-spatial features.

6.1.1 Full Trajectory Evolution

In this strategy all of the points of the trajectory are evolved, and after that process, they are evaluated. The number of parameters to evolve, at the same time, in this case is DoF $\cdot N$, where DoF is the number of joints involved in the movement, and N is the number of points of the trajectory. For example, for a spatial trajectory using 3 joints (DoF $= 3$) in a feature trajectory of 5 points ($N = 5$), the number of parameters to evolve is 15. **This strategy tends to make the algorithm converge slowly**, because the search space in this case usually becomes very large.

6.1.2 Individual Evolution

The second strategy consists in evolving and calculating the fitness for each feature point in the trajectory individually, instead of evolving and evaluating the feature trajectory as a whole. **This is valid for spatial trajectories**, where the joint parameters for reaching a point do not depend on previous points. In this case the number of different evolutions to be performed is equal to N, and each of these evolutions must evolve a number of parameters equal to DoF. **This second strategy can outperform the first one in time and fitness value**, mostly due to the smaller search space to be examined in each case.

6.1.3 Incrementally Evolved Trajectories

For features which have a **temporal dependency**, which is the case of many non-spatial features, the Individual Evolution strategy is not useful, as the action may have, and usually has, a dependency on previous points. For instance, when painting a wall, and using a CGDA encoding of the task, the percentage of painted wall depends on the wall that was painted before.

For these situations, we have developed a third strategy called Incrementally Evolved Trajectories. In this case, we also evolve each point individually. The first point is evolved individually. After it converges, we start to evolve the second point, but in

this case, the fitness evaluation is performed by sequentially executing both points (the previous first point and the current point). Once the second point converges, we start evolving the third point and for the evaluation we execute the three point trajectory. The same process is repeated for the remaining points. The pseudocode of this strategy is shown in Algorithm 1.

Algorithm 1 Incrementally Evolved Trajectories (IET)

```
1: procedure IET(X, ε)                ▷ X is the feature trajectory. ε is an error parameter.
2:     for i < numberOfPoints(X) do
3:         while fitness > ε do
4:             M_i ←evolve()
5:             f ←execute(M_[0:i])
6:             fitness ←compare(f, X_i)
7:         end while
8:     end for
9:     return M
10: end procedure
```

This technique offers some advantages with respect to the alternative strategies. First, it reduces the search space for each individual evolution, as it only evolves one point each time. Second, it also reduces the time necessary to converge, and improves fitness value of the whole trajectory. In the experimental section, we will show how it is able to approximate different object feature trajectories.

Moving from evolutionary computation, we realized we can leverage from physical interaction with the robot. Physical interaction can be translated into motor primitives, which are then used as pieces of a puzzle. When put in the correct order, primitives form a joint trajectory. Let us now extend on this idea.

6.2 Guided Motor Primitives

Our work on CGDA execution can be seen as a search of **inverse models** based on task goals i.e. compute the action policies that can generate a given effect. We benefit from human-robot interaction to create a library of Guided Motor Primitives (GMP). **These primitives are created by extracting segments of movements** from a random

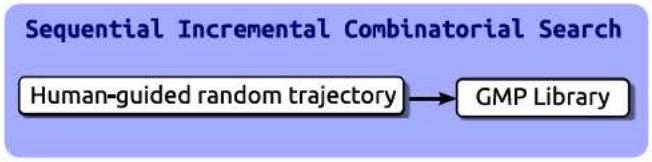

Figure 6.3: *Guided Motor Primitives block.*

human guidance with the robot. The process starts with the robot arm being moved randomly by the human (Figure 6.4).

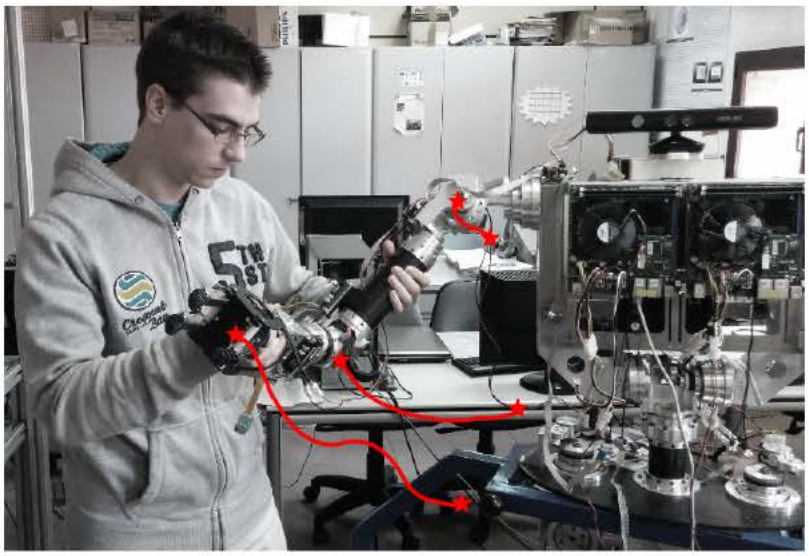

Figure 6.4: *Human guiding the robot in a random spatial trajectory. The red lines represent an example of the joints' spatial trajectories during the guiding.*

This guided exploration generates a joint trajectory in the joint space (an example using three joints is shown in Figure 6.5). The joint trajectory is split in small segments of τ seconds and all the segments are stored.

The process to select the most diverse primitives of the set starts with the first segment, which is automatically considered the first GMP. Then, the second segment is compared with the first. If it is different enough, it is considered a new GMP, if it is not different enough, we discard it and continue evaluating the next segment. Once more than one primitive is stored, the next segments are compared with all the stored primitives.

For comparison, the primitives are transformed to be relative to the origin of coordi-

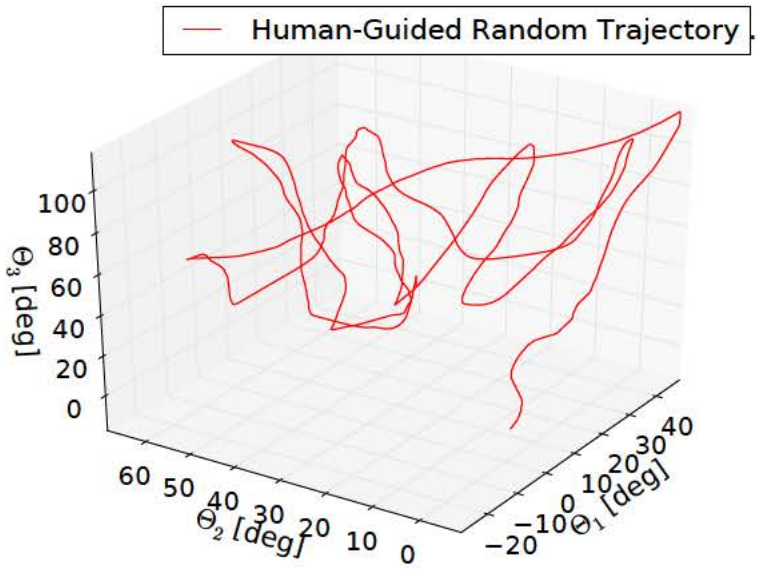

Figure 6.5: *This random joint trajectory involves 3 joints of the robot arm. It is generated when the human performs some random movements with the robot arm.*

nates, while conserving their shape. The comparison between two discretized segments is performed by applying the Euclidean distance over all pairs of points (one belonging to the primitive and the other one to the segment being evaluated) aligned in time. This analysis returns a single value J of discrepancy which is then used to store or reject the segment. If J is lower than a hand-crafted threshold ξ, the segment is discarded. An example of comparison between two random segments is shown in Figure 6.6.

For example, for the simulated humanoid robot experiment, explained in posterior chapters, a random guided movement was performed. After only 47 seconds of random human guidance, 94 segments were generated with $\tau = 0.5$ s, and 47 segments when selecting $\tau = 1$ s. In function of ξ (the threshold of similarity), these segments lead to sets of GMP with different number of primitives. Examples of GMP can be found in Figure 6.7.

One advantage of this library of GMP is its re-usability. The primitives are task-independent and can be applied to different situations. This affirmation will be demonstrated in the experimental section, where two experiments are performed with the same set of segments. Another advantage is that, in case a task cannot be accomplished with the current library of GMP, two mechanisms exist to increase its number. First, it is

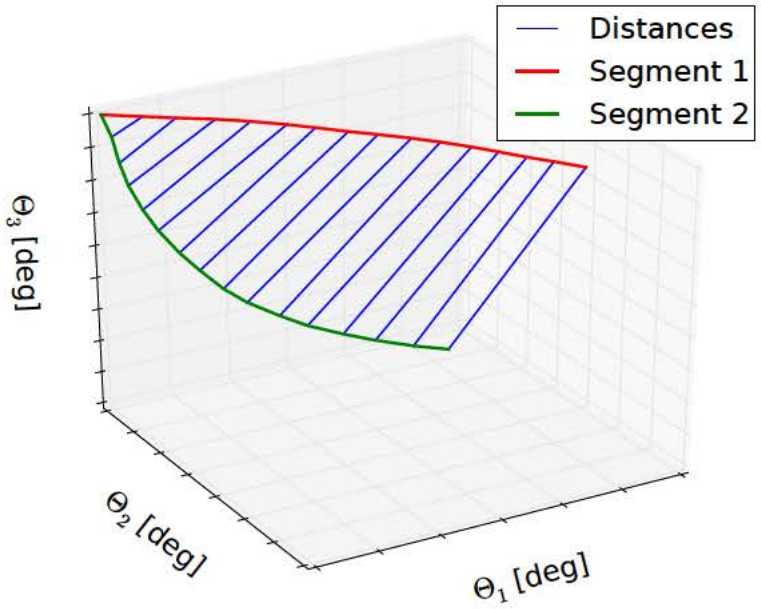

Figure 6.6: *An example of comparison between two random segments extracted from the previously shown joint trajectory. Both segments have a duration of $\tau = 0.5$ s and the blue lines represent the distances between pairs of points.*

possible to perform more human guided random interactions to obtain more primitives e.g. if new joints are available, a new human-guided interaction can reach currently unknown areas of the joint space leading to new primitives. A second mechanism is implicit in the threshold of similarity ξ. As it is a manually set parameter, its value may be decreased, incrementing the number of GMP.

6.2.1 Sequential Incremental Combinatorial Search

Once the GMP library has been created, we must focus on **how to combine sequentially the primitives for reproducing tasks encoded as CGDA**. The encoded features usually have a temporal dependency, where each step in the feature trajectory is dependent of the previous feature point. For instance, painting a surface a depends on how much of the surface has already been painted. For these kind of features, executing the primitives that will achieve each point of the feature trajectory independently is not feasible.

For this reason, a search method called Sequential Incremental Combinatorial Search (SICS) has been developed. SICS is essentially a **tree search algorithm** working in

Figure 6.7: *Several examples of guided motor primitives in the joint space. They were extracted from the joint trajectory shown in Figure 6.5.*

a breadth-first manner within trees with incremental depths. In our case, **each node represents a GMP**, and each edge has an associated value which represents the cost of traversing a path from the initial node to the current one. **Traversing a path is the result of executing the sequence of primitives (nodes) of the path**. The path cost is calculated in the feature space. It is the difference between the feature trajectory generated when the primitives of the path are executed in order, and the provided generalized feature trajectory. A summary of SICS can be found in Algorithm 2.

Two elements make SICS different from a standard breadth-first algorithm. First, **the level of depths of the tree are incremented if no solution is found** (a standard algorithm would stop here). The number of maximum levels is determined by the user, allowing even an infinite number (completeness is not guaranteed in this case). Second, as the **primitives are contained in the joint space but the evaluation is performed in the feature space**, the cost of sequentially executing two primitives can be lower than executing a single one. This fact prevents from discarding branches of the tree and forces an exhaustive evaluation through all nodes.

Algorithm 2 Sequential Incremental Combinatorial Search (SICS)

```
1: procedure SICS(X, ε, d)    ▷ X is the feature trajectory. ε is an error parameter. d is the maximum allowed depth.
2:     P ←()                  ▷ P is the sequence of primitives used.
3:     for i < numberOfPoints(X) do
4:         P ←search(P, ε, d)
5:     end for
6:     return P
7: end procedure
8: function SEARCH(P, ε, d)
9:     depth ← 1
10:    while depth < d do
11:        N ←breadthNodesIndexes(depth)
12:        for j < length(N) do
13:            C ←pathTo(N_j)
14:            f ←execute(P + C)
15:            cost ←compare(f, X_i)
16:            if cost < ε then
17:                P ←add(N_j)
18:                return P
19:            end if
20:        end for
21:        expandTree(N)
22:        depth ← depth + 1
23:    end while
24:    P ←addLowestCostPath()
25:    return P
26: end function
```

Let us outline the working mechanism of the algorithm. Assuming we provide a generalized feature trajectory as reference, and discretizing this trajectory in points, SICS sequentially searches for a solution for each feature point. A solution is considered valid when the generated joint trajectory produces objects features that are similar to

those of the reference. SICS first evaluates the costs of nodes belonging to the first level of depth of the tree of primitives. If the cost of the path to reach a given node is lower than a manually set parameter ε, it stops searching, stores the path, and continues searching a solution for the next point of the feature trajectory. If the cost is not lower than ε for any node of the level of depth, the tree is expanded one level of depth, and the costs of sequentially executing the two depth-level paths are analyzed (Figure 6.8).

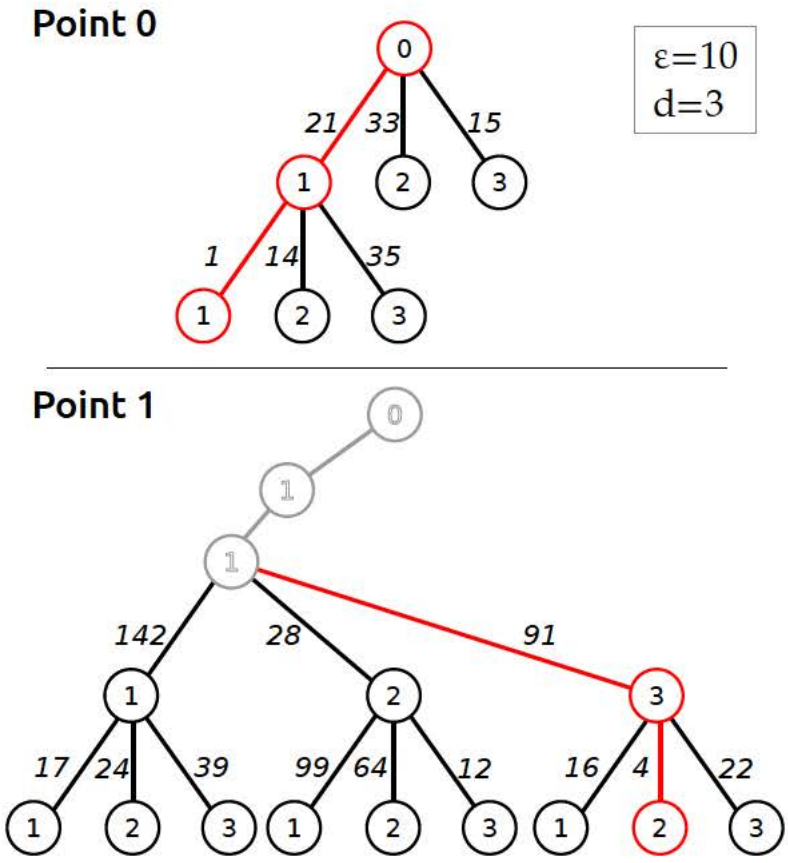

Figure 6.8: *SICS example for a two-point feature trajectory. For the first point, the nodes are expanded and a breadth-first search is performed until the cost of one of the nodes is lower than a threshold ε. If this occurs, the path to the node is stored and the process continues for the next point. If not, the tree is expanded one level of depth, and the search repeats.*

The same process of search and expansion is iteratively followed until a solution is found for all feature points (the error is lower than ε for all points), or the tree has expanded d times without finding an acceptable solution. The path cost is evaluated in the feature space by calculating the Euclidean distance between the discretized feature

trajectory that results from sequentially combining the primitives, and the discretized generalized feature trajectory.

These GMP have to be created by guiding the robot's arm through its free space for a while. As mentioned earlier, robots are mechanically stiff machines, with high reduction gears that hamper any physical interaction. Even a partial compensation of this friction to help users in the task is welcomed. This is the aim of the next section, to provide a help for users in robot learning scenarios.

6.2.2 Force-Sensorless Friction and Gravity Compensation

Figure 6.9: *Friction and Gravity Compensation block.*

To facilitate the creation of GMP, two simple controllers that compensate friction and gravity forces have been designed. **These controllers just aim to help users at guiding the robot in their movements**. More complex and accurate friction and gravity models can be found on the literature.

The controllers are related with the fundamental laws of dynamics for serial rigid multibody systems. Consider the Euler-Lagrange equations of motion of multibody rigid links in the robot joint space as:

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F_v\dot{q} + F_s\,\mathrm{sgn}(\dot{q}) + g(q) = u + \tau_{ext} \tag{6.1}$$

Where B is the inertia matrix, C represents the centrifugal and Coriolis forces, $F_v\dot{q}$ is the viscous friction torque, $F_s\,\mathrm{sgn}(\dot{q})$ is the Coulomb friction torque, F_v is the matrix of viscous friction coefficients, $g(q)$ is the gravity term, u is the actuation torque, and finally τ_{ext} is the torque originated by external forces. **An ideal friction and gravity compensator** could be expressed as a u with the following form:

$$u = F_v\dot{q} + F_s\,\mathrm{sgn}(\dot{q}) + g(q) \tag{6.2}$$

Due to the low speeds applied in robotics, the Coriolis and centrifugal forces C are negligible. Substituting (6.2) in (6.1):

$$B(q)\ddot{q} = \tau_{ext} \tag{6.3}$$

Which means that the mechanism would offer a resistance to external forces (e.g. pushing or pulling) equivalent only to its inertia. When applied to a robotic system, this controller would make the whole mechanism behave as if it were in free movement. Our controllers combine friction and gravity compensation terms to provide new forms of physical interaction with robots. Let us formally describe the equations governing the controllers. Let $g(q)$ be the term of gravity compensation, with q as the actual joint configuration. Let $\tau_f(q, \dot{q})$ be the term of friction compensation, where $\dot{q}$ is the joint angular velocity. Then, a generic friction and gravity compensation controller can be expressed as:

$$u = g(q) + \tau_f(q, \dot{q}) \tag{6.4}$$

A block scheme of this generic friction and gravity compensation controller can be seen in Figure 6.10. Let us now describe how the gravity and friction compensation terms can be determined.

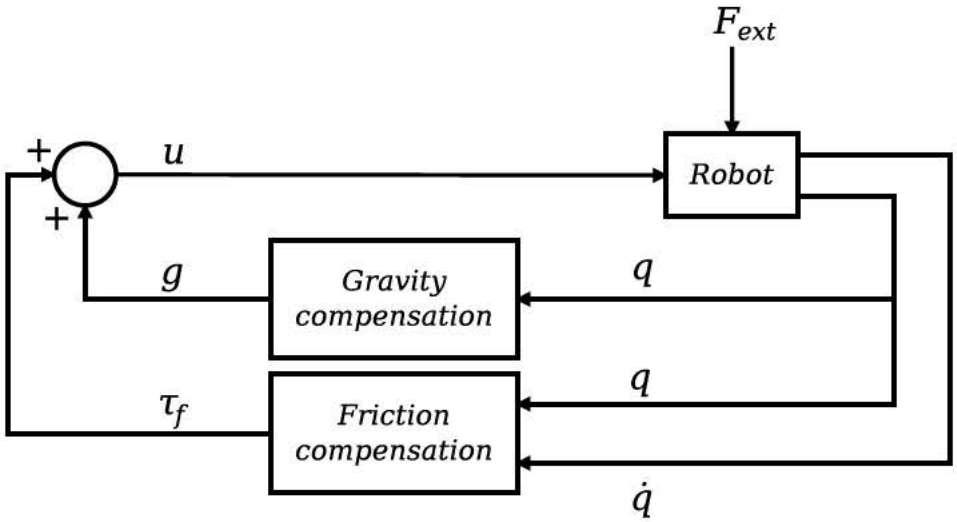

Figure 6.10: *Block scheme of friction and gravity compensation. In this type of control, there is no external reference, the robotic system is moved by the external perturbations caused by the user.*

Gravity Compensation

The potential energy of a robot, assuming rigid links and punctual masses, can be defined as (Sciavicco & Villani, 2009):

$$U = \sum_{i=1}^{n} (U_{l_i} + U_{m_i}) \tag{6.5}$$

Where U_{l_i} is the sum of potential energy contributions of each link, U_{m_i} is the contribution of each motor, and i is an index for each link or motor. The first term U_{l_i} is defined as:

$$U_{l_i} = -m_{l_i} g_0^T p_{l_i} \tag{6.6}$$

Where m_{l_i} is the mass of the center of masses of link i, g_0 is the gravity vector expressed in base frame (e.g. $g_0 = [0 \quad 0 \quad -9.81]^T$), and p_{l_i} is the set of coordinates of the center of masses of link i expressed in the base frame. Similarly, the motor contributions U_{m_i} are defined as:

$$U_{m_i} = -m_{m_i} g_0^T p_{m_i} \tag{6.7}$$

Substituting (6.6) and (6.7) in (6.5), U becomes:

$$U = -\sum_{i=1}^{n} (m_{l_i} g_0^T p_{l_i} + m_{m_i} g_0^T p_{m_i}) \tag{6.8}$$

Where p_{l_i} and p_{m_i} depend on the joint configuration q. **The torque $g(q)$ exerted by gravity can be computed as** (Luca & Panzieri, 1993):

$$g(q) = \frac{\partial U}{\partial q} \tag{6.9}$$

And is thus the torque required for gravity compensation. In the real world, determining the influence of each element in the potential energy equation is a non-trivial issue. For instance, the distinction between motor and link mass contribution is blurry, as the mass contribution between motors includes the parts of the motors located between the axes of rotation. This is the reason why we will use a simplified dynamic model of U. In this simplified model, the terms of link and motor contributions are mixed, and their masses are concentrated in the intermediate point between each pair

of axes of rotation. This dynamic model is commonly used in humanoid robot research, and is usually called 'mass concentrated model'.

Friction Compensation

The static friction forces, $F_v\dot{q} + F_s\,\mathrm{sgn}(\dot{q})$, from (6.1) can be compacted into a joint friction term, $\tau_{fj}(\dot{q})$. It can be computed with a **model-based identification procedure** inspired by (Virgala & Kelemen, 2013). Among the available friction models, they have assumed the one including Coulomb friction (initial opposing torque) and viscous friction (friction dependent on velocity). Their aim is to model the friction of an electric motor. The motion of an electric motor can be described as:

$$\tau_m - \tau_{fm}(\dot{\theta}) = J\ddot{\theta} \tag{6.10}$$

Where τ_m is the motor torque, $\tau_{fm}(\dot{\theta})$ is the motor friction torque, $\ddot{\theta}$ is the motor angular acceleration and J is the inertia of the motor. If the angular velocity $\dot{\theta}$ is stabilized, then $\ddot{\theta} = 0$, so the torque of the motor is used exclusively to compensate the friction:

$$\tau_m(t) = \tau_{fm}(\dot{\theta}) \tag{6.11}$$

Measuring the different velocities where the motor stabilizes for several torques applied, the stabilized velocities for these different torques can be plotted. The friction model selected by (Virgala & Kelemen, 2013) becomes a piecewise linear model:

$$\tau_{fm}(\dot{q}) = \begin{cases} \alpha_1\dot{\theta} + \beta_1 & : \ \dot{\theta} > 0 \\ \alpha_2\dot{\theta} - \beta_2 & : \ \dot{\theta} < 0 \end{cases} \tag{6.12}$$

Where model parameters α and β are obtained by linear regression on the plot (Figure 6.11).

In the original procedure, they measure the motor velocity $\dot{\theta}$ in isolation. In our proposed identification procedure, we measure the velocity $\dot{q}$ with the motor within the robot, including the gearboxes and the mechanical structure. Modeling each part independently (motor, gearbox, structure, construction) would result in intractable combinations of models to be evaluated and coordinated, specially for many DoF. Our assumptions are the following:

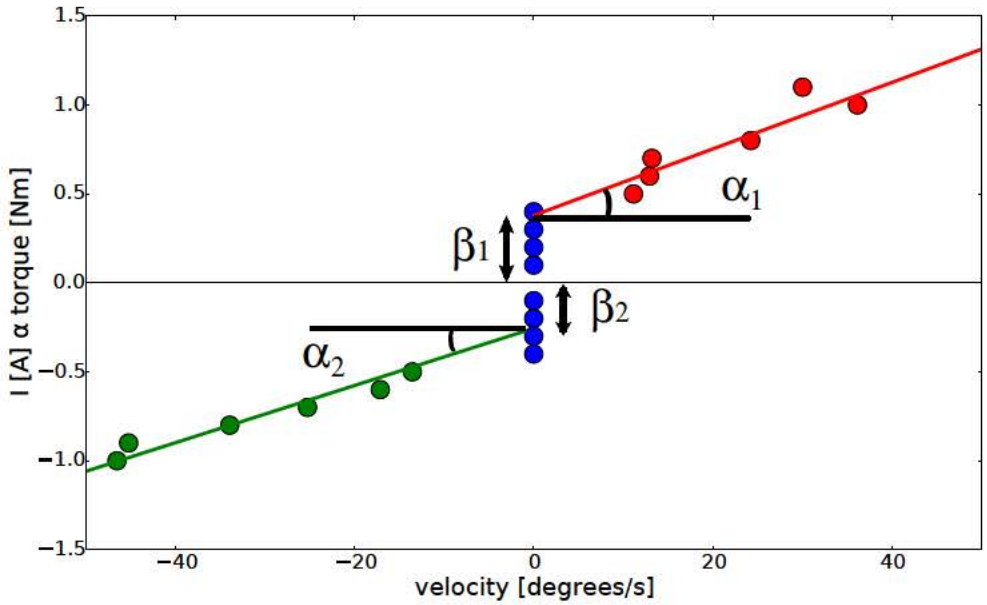

Figure 6.11: *This friction model includes Coulomb friction and viscous friction. The variables α and β represent the parameters of the linear model assumed.*

- We use **joint velocity $\dot{q}$ instead of motor velocity $\dot{\theta}$.** The joint friction model selected becomes the following piecewise linear model:

$$\tau_{fj}(\dot{q}) = \begin{cases} \alpha_1 \dot{q} + \beta_1 & : \dot{q} > 0 \\ \alpha_2 \dot{q} - \beta_2 & : \dot{q} < 0 \end{cases} \tag{6.13}$$

  Where $\tau_{fj}(\dot{q})$ is the torque necessary to compensate the friction generated in function of the joint angular velocity $\dot{q}$.

- We assume that the **motors have a symmetrical behavior** i.e. they oppose to movement in both directions with the same strength. Therefore, $\alpha_1 = \alpha_2$ and $\beta_1 = \beta_2$.

- When applying constant torques, we **limit the time given for velocity stabilization** due to the mechanical constraints of the robot joints.

- As we have to deal with gravity forces, which may influence friction, we add an additional term when opposing gravity.

Robot joints have mechanical constraints, so there is a limit in the time the joint velocity can be recorded. This time may not be enough for the velocity to stabilize. In these cases, the velocity achieved before reaching the joint limit must be used instead of the stabilized velocity. This causes a steeper slope of the posterior linear regression. The final parameters of the linear regression should be further adjusted in these cases.

As stated in our final assumption, we add a term in addition to $\tau_{fj}(\dot{q})$. We assume that an additional mechanical friction is generated in the motor axle and gearbox due to gravity. This is the reason why we have added a term τ_{fg} dependent on the joint position and the velocity:

$$\tau_{fg} = f(q, \dot{q}) \tag{6.14}$$

The term $\tau_{fg}(q, \dot{q})$ is purely experimental, as it depends on the mechanical design and construction of the robot. In our model, we only add this term when the gravity opposes the direction movement of the arm. To see whether the gravity is in favor or against this movement, the variation of the potential energy U can be used. When $\Delta U > 0$, the movement is against gravity. The final friction compensator can be expressed as:

$$\tau_f(q, \dot{q}) = \begin{cases} \tau_{fj}(\dot{q}) + \tau_{fg}(q, \dot{q}) & : \ \Delta U > 0 \\ \tau_{fj}(\dot{q}) & : \ \Delta U < 0 \end{cases} \tag{6.15}$$

Different applications may require different behaviors of the robot. Hence, two controllers have been derived from the generic friction and gravity compensation controller (6.4).

Low-Friction Zero-Gravity controller (LFZG) This controller can improve the physical interaction with robots. In this controller, a new parameter ξ has been incorporated. This parameter attenuates the influence of the friction compensation on the system. Introducing ξ in the controller, it becomes:

$$u = g(q) + \xi \tau_f(q, \dot{q}) \tag{6.16}$$

By setting $0 < \xi < 1$, this controller allows the robot to move easily, without effort, but eventually stopping due to the low friction. This controller can be useful in paradigms such as keyframe demonstration and PbD, where there is a direct

physical contact with the robot. For instance, when aiming to record a task using keyframe demonstration, different robot configurations must be recorded. In many cases, a demonstrator may have to use both hands to move a single robot joint, due to its individual friction. Therefore, in robots with many DoF, it can be difficult to physically move the robot between the different desired configurations. Using our controller, one has to simply push the robot in the desired direction, and stop it when desired. The attenuated friction serves as an aid for stopping at the desired target keyframes.

Zero-Friction Zero-Gravity controller (ZFZG) This controller, when applied to all the joints, ideally makes the robot move as if only the external dynamic forces and inertia would modify the motion. To achieve this behavior, we can use the generic friction and gravity compensation controller (6.4):

$$u = g(q) + \tau_f(q, \dot{q}) \tag{6.17}$$

This controller, which is a special case of LFZG, can be useful for situations where it is interesting to simulate gravity-free conditions. A robotic platform using this control could be employed to test how devices would behave in complete absence of friction and gravity. Obviously, this behavior would only be reached if the controller is able to compensate perfectly both gravity and friction. Usually, such accuracy will not be achieved, and the controller will behave as a LFZG with a ξ close to one.

The theoretical descriptions and the mathematical formulations of the research of this thesis end with this chapter. The next chapters will focus on experimental validation, analysis of the results and their discussion.

Chapter 7

Experiments

Many different research topics and novel techniques have been presented in this thesis. In this chapter, each novelty is analyzed in its own section.

1. Demonstration and feature selection is tested by recording several demonstrations of a task using the robot's sensor devices. This information is subject of study. See Section 7.1.

2. CGDA recognition is evaluated. Despite recognition is implicitly evaluated in the execution section, it is also tested independently. See Section 7.2.

3. CGDA using evolutionary computation is tested in a simulated scenario. Two tasks objectives are evaluated, using all the evolutionary strategies presented before. See Section 7.3.

4. To create a Guided Motor Primitives library, a human has guided a humanoid robot's arm in random movements. The generated primitives are evaluated in a simulated scenario. GMP and its tree search algorithm substitute evolutionary computation, as optimization technique, in this experiment. See Section 7.4.

5. Finally, friction and gravity compensation is applied to 1 DoF of the robot arm to test the effectiveness of the approach. See Section 7.5.

7.1 Demonstration and Feature Selection

The first goal of this experiment is to distinguish correct and incorrect demonstrations of a given task. The second goal is to distinguish between relevant and irrelevant features for the task.

The task flows as follows: a human grasps the the green cylinder with one hand. Then, tracing an arc, he places it on the top of the red cylinder. The recording is manually started and stopped. The red cylinder is not moved in any demonstration, while the green cylinder starts in a different position, on the table, for each demonstration (Figure 7.1).

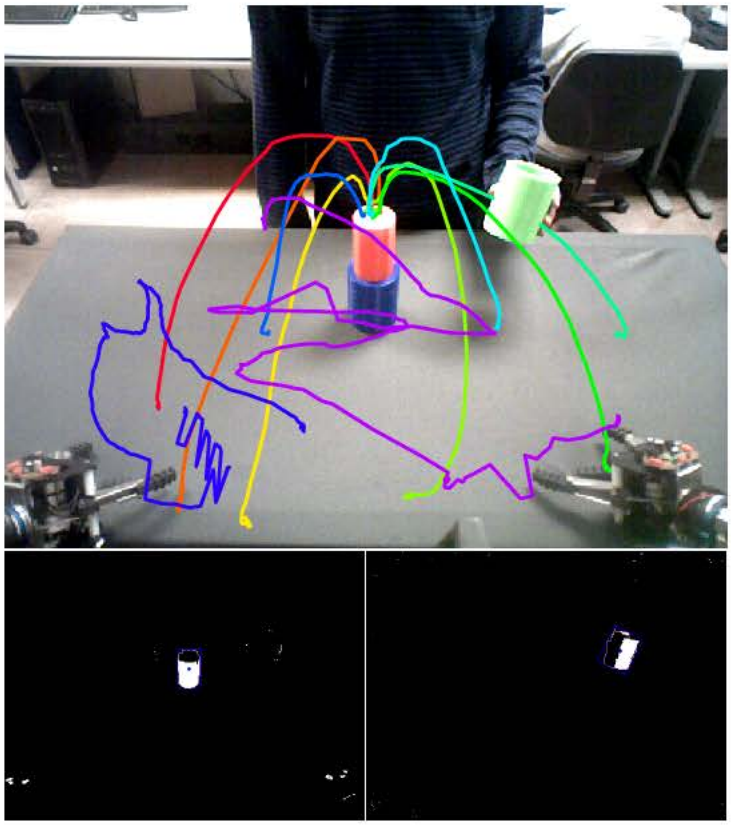

Figure 7.1: *User demonstrations from TEO's perspective; red segmentation on the bottom left, and green segmentation on the bottom right.*

We have used the full-size humanoid robot TEO (Martínez et al., 2012). TEO's head is equipped with an ASUS Xtion PRO LIVE set to provide 640×480 RGB and depth streams at 30 fps. The red and the green object are segmented by color, as in the lower

part of the Figure 7.1. The following 13 scalar features are extracted in a periodic 40 ms loop:

- Centroid absolute position (red object x_1, y_1, z_1 and green object x_2, y_2, z_2)
- Centroid relative position (the difference between the centroid absolute positions $x_1 - x_2$, $y_1 - y_2$, $z_1 - z_2$)
- Absolute values of the previous values ($|x_1 - x_2|$, $|y_1 - y_2|$, $|z_1 - z_2|$)
- Euclidean distance between the red and the green object $dist(X_1, X_2)$

We recorded 10 demonstrations of different durations, performing 8 of them correctly, and performing the last 2 incorrectly. The red object is not moved in any of the correct demonstrations, but it is moved in the incorrect ones. The green object approaches the red object from different angles in the correct demonstrations, and is moved randomly in the incorrect ones. As humans, with this context information, we consider that the relevant demonstrations (those to be kept), are those which are similar to most of the other, so **we would discard the last two demonstrations**.

Regarding the features, we consider that the **features that must be discarded** are: x_2, y_2, $x_1 - x_2$ and $y_1 - y_2$, which are **those dependent on the initial position of the green object**. The rest of the variables should not be discarded: variables involving z remain constant across demonstrations, absolute value differences of the objects tend to the same values when sign is removed, and finally, the Euclidean distance always decreases for all demonstrations, as the green object is always moved closer to the red one.

7.1.1 Demonstration Selector

The spatial movements of the red object for all the demonstrations can be seen in Figure 7.2. As the robot is fixed during the demonstrations, we can plot all the demonstrations in the same space, to compare their behavior.

As can be seen in the image, there is a concentration of lines in the middle (the object remains unmoved). Additionally, there are two lines (blue and purple) which deviate from the rest. These are the incorrectly performed demonstrations. The yellow line remains centered most of the time, except for one deviated point (due to a sensor failure). Figure 7.3 depicts the spatial movements of the red object for all demonstrations.

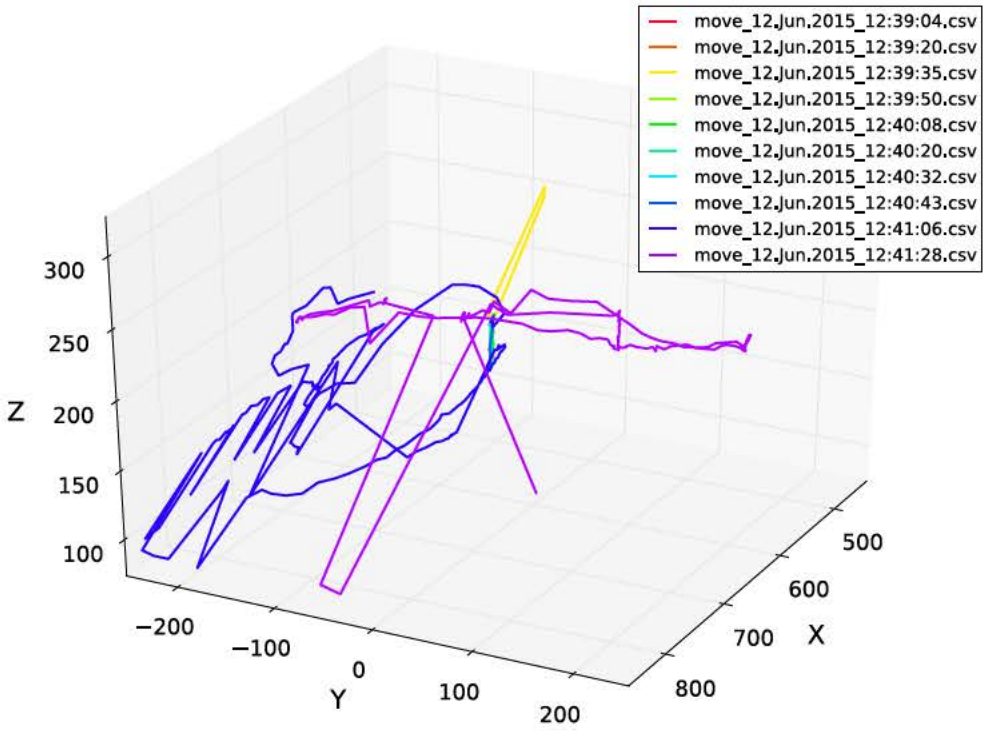

Figure 7.2: *Red object centroid coordinates throughout time.*

In this case, for the correct demonstrations, the green object converges to the top of the red object. In the incorrect demonstrations (blue and purple), it does not converge to this point. We have overlayed the green object movement into the robot point of view, to give the reader an understanding of the plots (Figure 7.1). Dissimilarity Mapping Filtering process has been applied to these demonstration. The selection of algorithms in this case is: Dynamic Time Warping (Dissimilarity), Summing columns or Summing rows (Mapping), z-score (Filtering).

We can plot the z-scores of the demonstrations. In the first figure (Figure 7.4) we have applied the *summing columns* preprocessing, and treated the data without normalization.

We have also tested the *summing rows* preprocessing (Figure 7.5), also without normalization.

In both cases, **the discarded demonstrations are those that were incorrectly performed**, when using a threshold of $\alpha = 0.5$. Despite of the threshold used, the process has been able to separate the correct and the incorrect demonstrations.

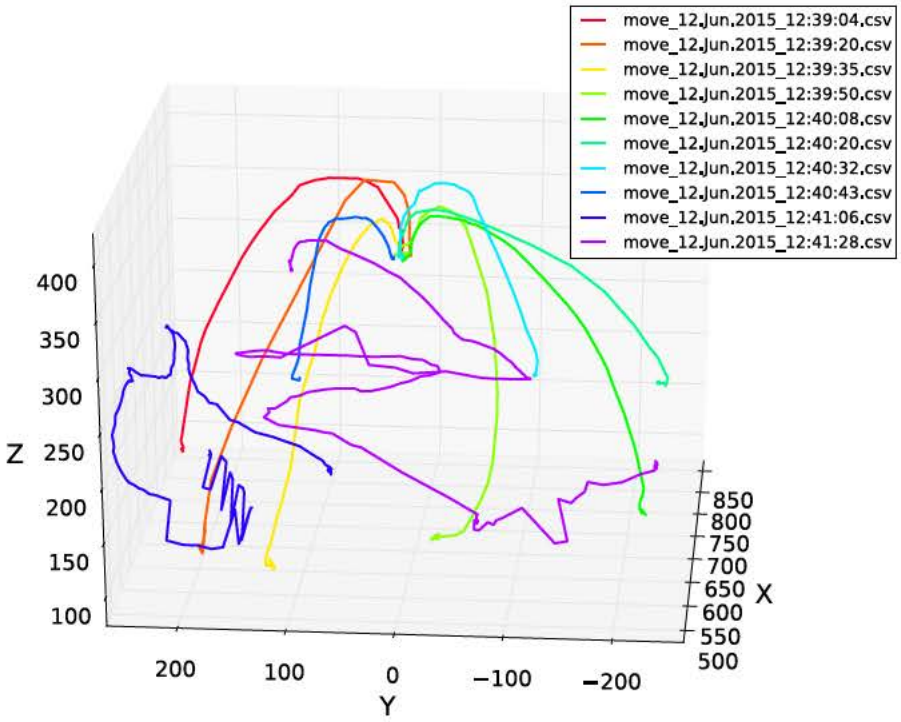

Figure 7.3: *Green object centroid coordinates throughout time.*

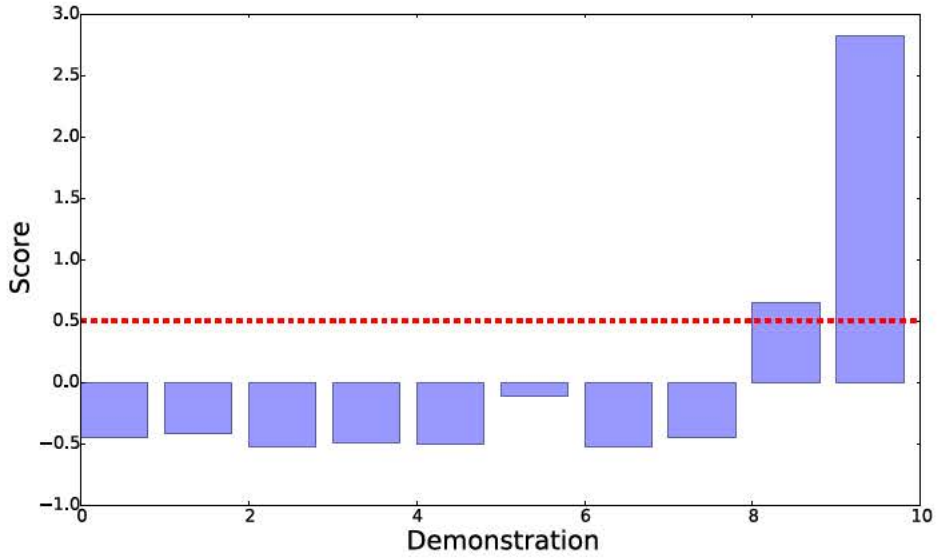

Figure 7.4: *Demonstration selection's z-scores without data normalization, preprocessed by summing columns, setting the threshold to* $\alpha = 0.5$.

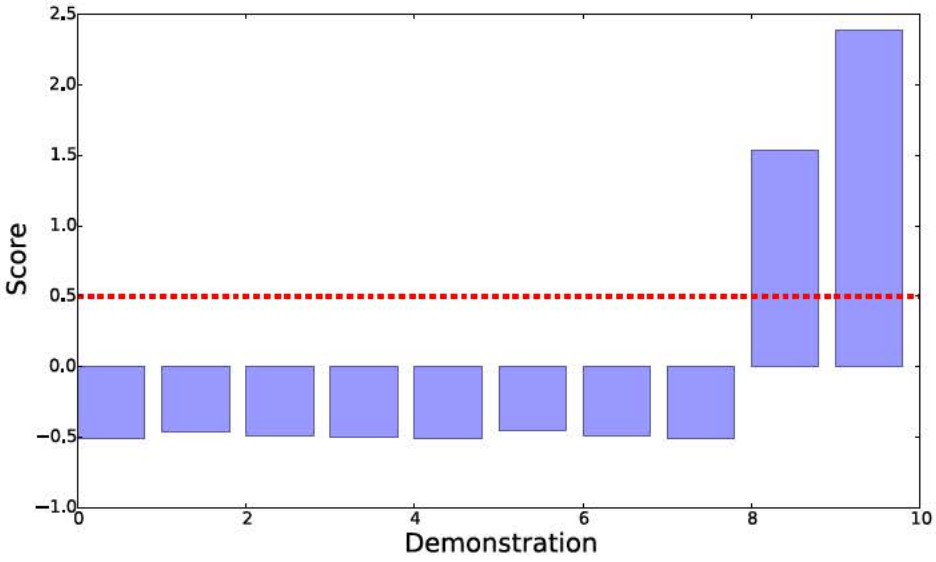

Figure 7.5: *Demonstration selection's z-scores without data normalization, preprocessed by summing rows, setting the threshold to* $\alpha = 0.5$.

7.1.2 Feature Selector

Once the two incorrect demonstrations were discarded, we applied the feature selection algorithm on the remaining ones. The selection of algorithms in this case is: Dynamic Time Warping (Dissimilarity), Summing columns (Mapping), z-score (Filtering). The z-scores results for the feature selection can be seen in Figure 7.6. The discarded features are x_2, y_2, $x_1 - x_2$ and $y_1 - y_2$. This result agrees with our previous expectation. The data was treated without normalization.

Without normalization, **the discarded features are those already expected**. While this decision design has worked in this experiment, other experimental situations may require a data normalization. Several normalization methods were applied to the experiment raw data, and feature selection was applied again for each normalization method used. As a summary, we have gathered the correspondent z-scores, when data is normalized with several techniques, in Table 7.1.

As can be deduced from the table, even when using different normalization techniques, DMF can approximately discard the correct features in all of the cases. Notice that we have reused the threshold value $\alpha = 0.5$. However, using different threshold values for different normalization techniques can boost the results in most of them.

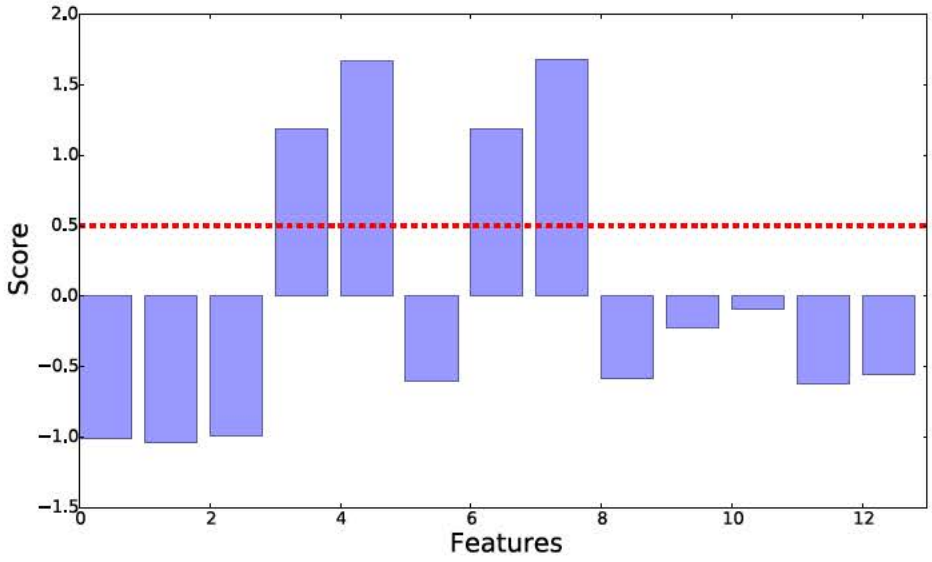

Figure 7.6: *Feature selection's z-scores without data normalization, setting the threshold to* $\alpha = 0.5$.

Table 7.1: *z-scores with normalized data. A green cell means the feature is accepted. A red cell means the feature is discarded. Threshold is set to* $\alpha = 0.5$.

	MinMax	*Standardize*	*Whole Exp.*	*Phys. Limits*
$\mathbf{x_1}$	-0.004	0.301	-0.956	-1.030
$\mathbf{y_1}$	0.743	0.650	-0.889	-1.077
$\mathbf{z_1}$	-0.657	-1.080	-0.929	-0.985
$\mathbf{x_2}$	1.223	1.209	-0.37	2.396
$\mathbf{y_2}$	1.439	1.334	1.874	1.570
$\mathbf{z_2}$	-1.080	-1.010	-0.728	-0.226
$\mathbf{x_1 - x_2}$	1.166	1.201	1.670	0.662
$\mathbf{y_1 - y_2}$	1.435	1.334	1.679	0.246
$\mathbf{z_1 - z_2}$	-1.088	-1.032	-0.257	-0.637
$\mathbf{\lvert x_1 - x_2 \rvert}$	-0.782	-0.891	0.004	0.204
$\mathbf{\lvert y_1 - y_2 \rvert}$	-0.917	-0.977	-0.041	-0.152
$\mathbf{\lvert z_1 - z_2 \rvert}$	-0.484	-0.034	-0.440	-0.258
Eucl.Dist	-0.992	-0.996	-0.609	-0.711

7.2 CGDA Recognition

Prior to including the recognition module into an execution experiment, it must be tested separately. In this section, we also aim to evaluate whether actions can be distinguished by their effects on objects.

7.2.1 Object Feature Trajectory Recognition

The first experiment framework consists in a Kinect camera pointed at a desktop tracking a colored marker. A demonstrator performed several action using the marker in front of the camera. Using YARP software (Metta, Fitzpatrick, & Natale, 2006), we connected the camera input with a computer vision library (Morante, 2012) to measure marker features. Four different basic actions were selected. We have given names to each basic action for simplicity in explanation, but semantics is not used in the process. The actions involve spatial movements (MOVE, ROTATE, CLEAN) and color changes (PAINT), as seen in Figure 7.7.

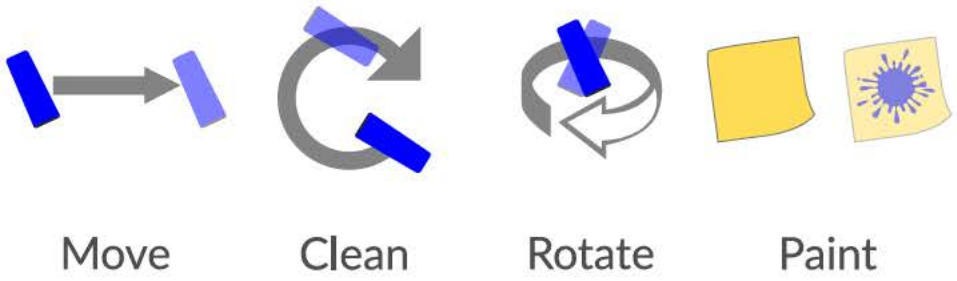

Figure 7.7: *Graphical representation of the actions performed in this experiment.*

Each action can be described as follows:

- MOVE: Marker displacement of 30 cm in one straight direction.
- ROTATE: Rotation of 90 degrees over the Center-of-Mass, on one axis.
- CLEAN: Keeping orientation fixed, movement of Center-of-Mass over the perimeter of a circle of 30 cm of diameter (1 revolution).
- PAINT: Keeping its spatial coordinates fixed, the marker is painted in a different color with a marking pen, until almost all of the area is covered.

We want to test our algorithms with small sets of repetitions, in order to prove its performance when there are few repetitions available. Seven repetitions of each basic action are recorded. **Six of the repetitions are used to generate one generalized action, similar to common practices of PbD** (Calinon et al., 2007). The final repetition of each set is used as a test action to be recognized. The tracked object features are: spatial location (X,Y,Z), area, HSV color (hue, value, saturation) and angle. Each test action is sent to the DTW recognition process, which compares it to each of the previously generated generalized actions. Results are shown in Table 7.2.

Table 7.2: *DTW cost matrix: test actions (lower case) vs. generalized trajectories (upper case). Bold numbers represent best values (minimum discrepancy).*

	MOVE	*ROTATE*	*CLEAN*	*PAINT*
move	**229**	332059	290334	552055
rotate	389021	**7606**	325211	694049
clean	402555	304669	**1724**	44259
paint	497152	671078	25896	**1277**

We perform the same comparison using only spatial features, and ignoring the others. The results can be seen in Table 7.3.

Table 7.3: *DTW cost matrix: reduced test actions (lower case) vs. reduced generalized trajectories (upper case). Bold numbers represent best values (minimum discrepancy). Only spatial features are used.*

	MOVE	*ROTATE*	*CLEAN*	*PAINT*
move	**8.15**	10251.49	11428.03	4888.67
rotate	12836.77	**8.94**	10035.21	284.87
clean	12252.87	8977.23	**13.46**	5175.71
paint	4728.14	135.77	5021.54	**14.33**

These results show that the measure of discrepancy, using DTW, between similar feature trajectories is lower than between different actions. This makes the recognition algorithm to correctly associate all the test trajectories with their set, in both cases. Tables 7.2 and 7.3 show the influence of additional dimensions on the comparison. As more dimensions are used, the quality of the results decays. Deeper analysis is needed

to check how other comparisons techniques behave in this situation.

7.2.2 Cartesian Space Trajectory Recognition

To check whether the CGDA approach is useful, during the previous experiment, we also measured the Cartesian positions (X,Y,Z) of the human demonstrator's arm joints: hand tip, wrist, elbow and shoulder. This data was incorporated in this experiment to highlight the differences in feature and Cartesian comparison. The demonstrator did not care to perform the actions in a specific kinematic way, or even a consistent one, between repetitions. The only aim was to accomplish the action description. When comparing test and generalized Cartesian actions (following the same scheme as previously), we obtain Table 7.4.

Table 7.4: *DTW cost matrix: Cartesian test actions (lower case) vs. Cartesian generalized trajectories (upper case). Bold numbers represent best values (minimum discrepancy).*

	MOVE	*ROTATE*	*CLEAN*	*PAINT*
move	**0.0032**	0.1594	0.0448	0.1031
rotate	0.1563	**0.0123**	0.0939	0.0430
clean	0.0234	0.0323	**0.0003**	0.0371
paint	0.0486	0.0148	0.0300	**0.0004**

As shown in Table 7.4, in this case, the system also recognizes the actions correctly, but the score differences are lower. In Table 7.2, the correct answer is 1 to 3 orders of magnitude lower, while in Table 7.4 results are all quite similar. This proves that enabling CGDA, we are allowing the demonstrator to focus on task completion, rather than focusing on the kinematic consistency.

7.3 CGDA Execution by Evolutionary Computation

Once we are confident about the recognition module, an execution experiment, integrating recognition, can be performed. In this section, two experiments have been carried out. One of them involves executing a goal-directed action, which goal is to follow a spatial trajectory of an object. This experiment would be normally tackled by computing the inverse kinematics of the robot arm. The second experiment tries to

achieve a change of color of a simulated wall. The goal variable is the percentage which has been painted. It is an adequate example of a goal-directed action where kinematic information is not provided to the robot.

Despite it is not the aim of the thesis to establish a rigid method to reproduce an encoded CGDA, we have developed a simple experiment to check its feasibility. We have used Evolutionary Computation (EC) and a simulated model of a humanoid robot, as can be seen in Figure 7.8. We set a generalized trajectory as the reference. Evolutionary Computation performs joint parameter evolution until convergence, starting from an initial random robot joint position trajectory.

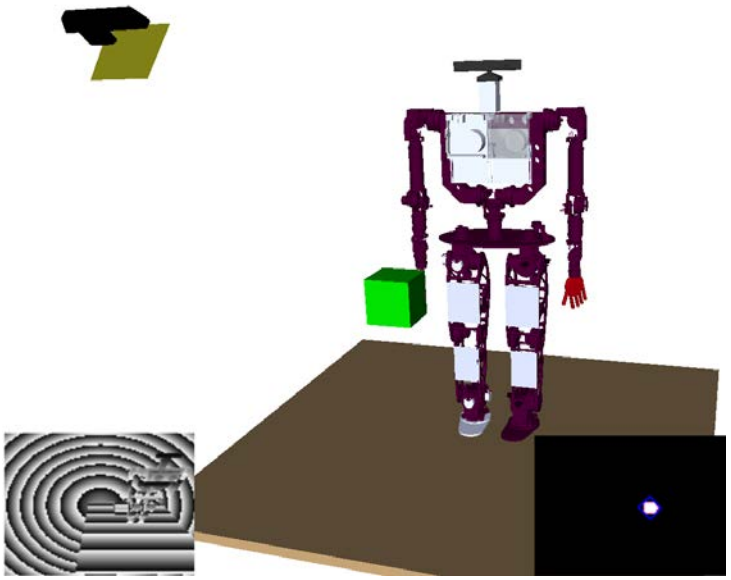

Figure 7.8: *The picture shows the experimental scenario with the robot, the object (green) and the Kinect camera. The bottom left square is the Kinect depth map and the bottom right square shows the color segmentation of the object.*

EC performs a steady state selection algorithm (few individuals are replaced after the selection and crossover process). In our implementation, EC uses a "Floating Point" genotype which is simply a vector of floating-point numbers which evolve until convergence. The following is a summary of the operator details:

1. Selection: A tournament is performed between 3 random individuals. Their fitness values are compared and winners are selected for crossover.

2. Crossover: Selection winners are crossed and their offspring substitute the worst individuals from the previous tournament.

3. Mutation: Finally, each child may be mutated with a 70% of probability.

Several strategies have been developed for the experiments. All of them use EC as optimization algorithm, but they differ on the strategy.

7.3.1 Executing a Spatial Task: Cleaning

The target of the task is to execute a generalized trajectory extracted from a set of repetitions. This trajectory contains only spatial features (X,Y,Z) where the object tracked is in one of the robot's hands. The action is the previously named as CLEAN. Due to the spatial encoding of the task, only the full evolution and the individual evolution strategy have been tested.

Full Trajectory Evolution

The generalized trajectory has a number of points that must be reached. To reduce the number of parameters to evolve, only 3 joints of one of the robot's arms are used. Fitness is evaluated when a full evolved joint trajectory is executed, by analyzing the features of a marker in the robot hand. The generalized goal action and the measured one are compared using the DTW recognition, and the score of discrepancy is used as the fitness value to minimize.

The termination condition is set to ten generations without improvement in the fitness value. The fitness evolution of our experiment can be found in Figure 7.9.

EC evolves 30 values (3 joint positions, times 10 timesteps) at the same time. Fitness is evaluated when the joint position trajectory is executed, by analyzing object features using a simulated Kinect camera in the environment. The reference action and the measured one are compared using CGDA recognition, and the score of discrepancy is used as the fitness value to minimize. We have chosen CLEAN as the action to be executed, and the object features measured are purely spatial ones (X, Y, Z). After convergence, the winner action is executed. Its performance compared to the generalized reference is depicted in Figure 7.10.

The resulting trajectories, when executed on the robot, are not human natural movements. This is an expected behavior, as the aim of the experiment was to replicate object

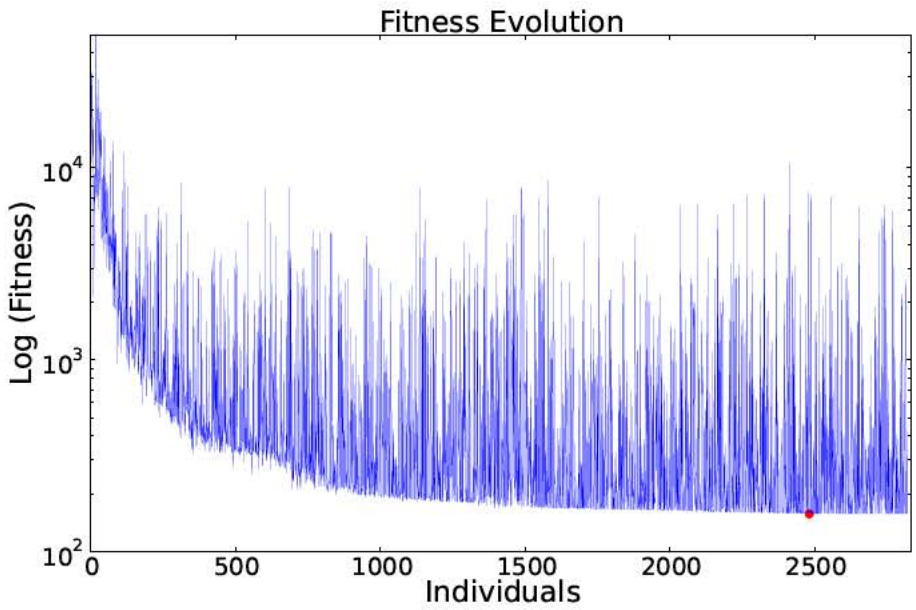

Figure 7.9: *Fitness value through evolution. The red point is the minimum value achieved by evolution.*

states during execution, and not the performance of human-like movements. In subsequent sections we will focus on improving the precision of the execution.

Individual Evolution

In this case, each point of the generalized trajectory is the target value for an individually evolved strategy. An evolved point is considered valid if it has an error lower than 20 mm to the reference point. Once the evolution has finished, the evolved action is executed. Unlike the previous case, the error is now measured using Euclidean distance. For a nine point trajectory, EC took an average 53 ± 12 s to calculate the whole trajectory (for 30 trials in an average computer), approximately 6 s to calculate each point. A comparison between the generalized trajectory and the evolved one can be found in Figure 7.11.

The evolved trajectories are bounded within the 20 mm limit with respect to the reference. The whole sets of trajectories (original repetitions, generalization and evolved) represented in a 3 dimensional figure, can be seen in Figure 7.12.

These results outperform those of the previous strategy, and also can be improved by imposing an acceptable error lower then 20 mm, but it would require more time to

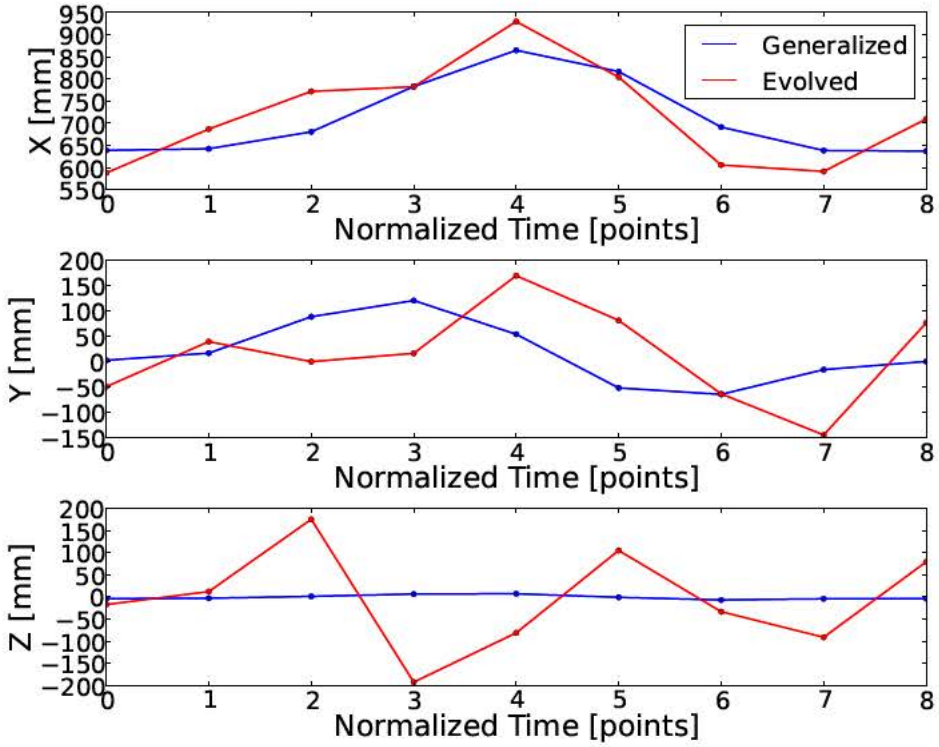

Figure 7.10: *Unidimensional temporal plots of generalized reference (blue), and the object feature space trajectory from executing the EC winner joint position trajectory (red). The Z dimension gives the worst results. The system was not able to reduce the error in this dimension.*

converge. The evolved full trajectory is depicted in Figure 7.13.

7.3.2 Executing a Non-Spatial Task: Painting

This experiment's target is to paint a wall. The painting process is encoded as a CGDA feature trajectory, but this time the object tracked is the wall. The experiment is performed in simulation with a model of the humanoid robot TEO (Martínez et al., 2012).

We want to make the humanoid robot paint a wall, without teaching it the necessary motor parameters. In the simulated scenario, there is a wall in front of the humanoid robot, composed by 16 small squares, all with the same color. The final action objective is to change the color of all the squares. The generalized action is created synthetically

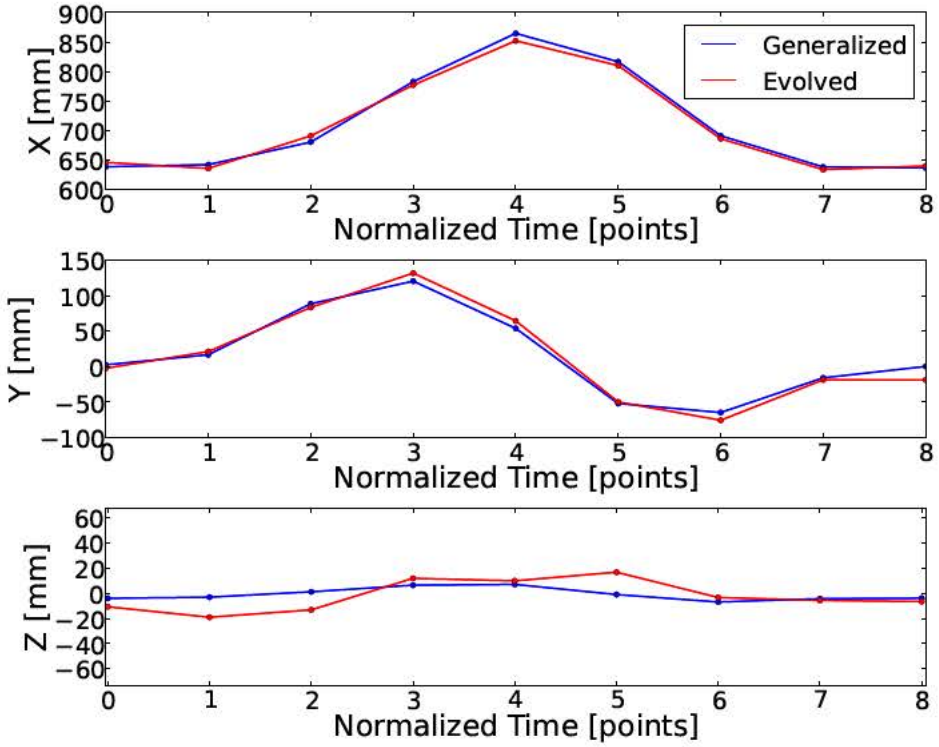

Figure 7.11: *Unidimensional temporal plots of the generalized action, and the trajectory obtained by executing the EC winner joint parameters. The trajectories are quite similar for all cases. In the case of Z dimension, which may look like a less well-defined result, it is important to check its units. In none of the points the error is bigger than 20 mm.*

and it has only one feature, which represents the percentage of painted wall. We force the percentage of painted wall to increase constantly with time, in 16 steps. In this case, due to the temporal dependency of the encoded feature, it is only feasible to test the incrementally evolved trajectory strategy.

Incrementally Evolved Trajectories

The generalized trajectory is used as the reference and EC is used to perform joint parameter evolution. In this case, every time the robot's hand gets closer than 120 mm to a square, this square changes its color. This is a simplification to emulate the real action of sliding a paintbrush over a surface. Fitness is evaluated when a joint trajectory

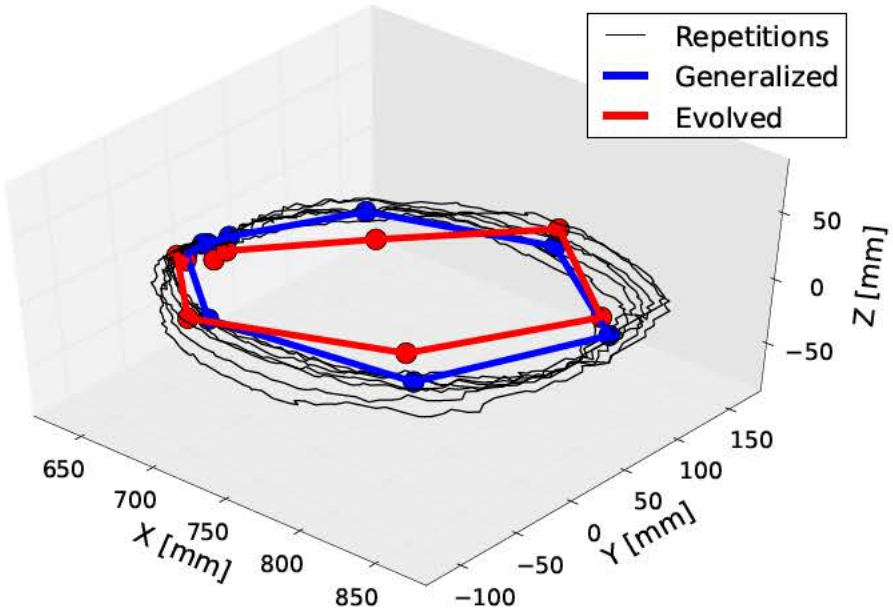

Figure 7.12: *Plot representing a three feature trajectory. Black lines are training action repetitions. The blue line is the generalization extracted from all the repetitions. The red line is the evolved trajectory. Note that none of the points of the evolved trajectory has an error bigger than 20 mm with respect to its reference (the generalized).*

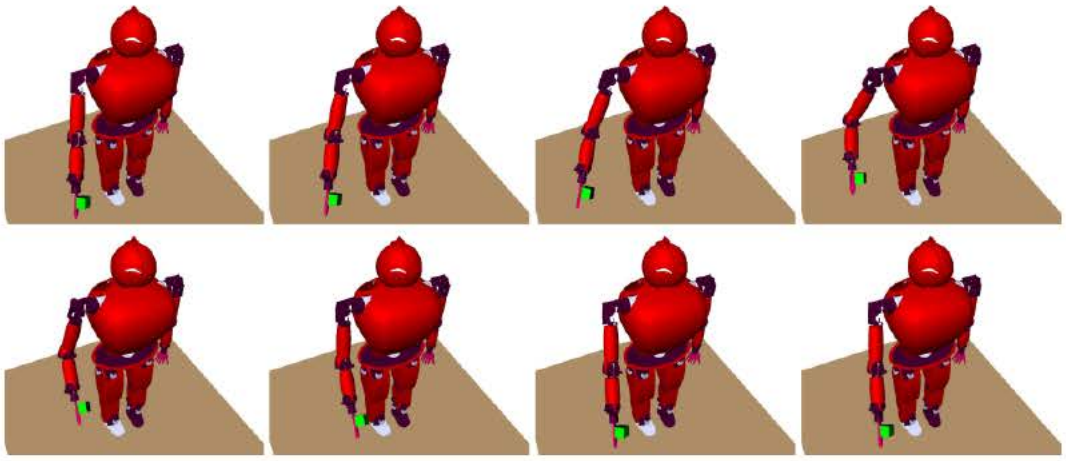

Figure 7.13: *Action execution for a CLEAN command. Despite the trajectory is purely spatial, the objective of the evolution and the execution is to perform a feature trajectory obtained from the object tracker.*

is executed, by analyzing wall features. For a trajectory composed by 17 points, EC took 258 ± 7 s to calculate the whole trajectory (as an average of 30 trials in an average

computer). It is important to notice that, as we are using the Incrementally Evolved Trajectories, the fitness evaluation gets more complex for each following point, as can be seen in Figure 7.14.

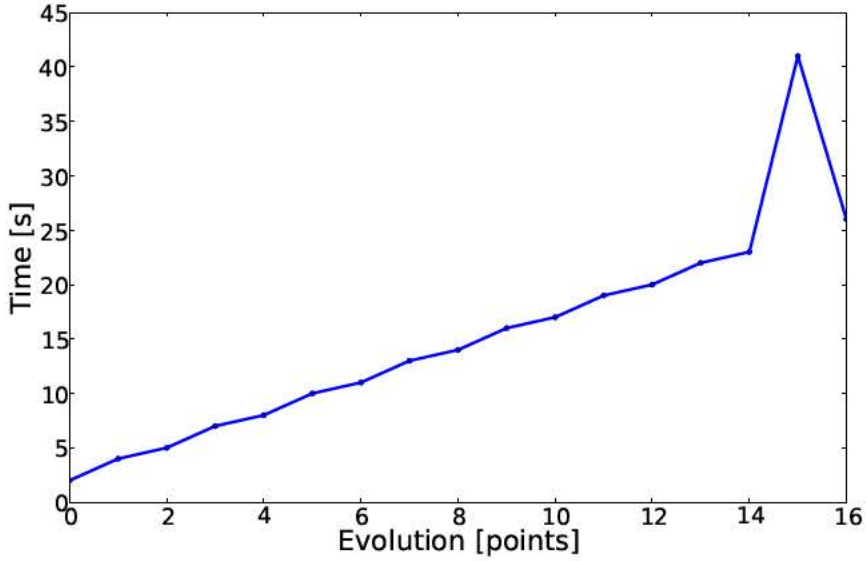

Figure 7.14: *In this chart the time taken to compute each point is shown. Except for one final point (where the valid space is already very restricted) the time of computation increases linearly.*

Each point becomes more difficult to compute than the previous one, because every time a point changes the color of a square it reduces the next point valid space. The drastic increment in the 15th point is a possible effect that may appear in restricted valid spaces. As the squares are changing their color, the valid space for next computations is reduced, rising the probabilities of longer operation times. A comparison between the generalized trajectory and the evolved one can be found in Figure 7.15.

The trajectory obtained in this case is highly acceptable. One of the reasons behind the good convergence is the use of the incremental evolution strategy, as the evolutionary algorithm only has to focus on one fitness each time. One theoretical disadvantage of the strategy employed is that it may accumulate error of previous points in following evolutions. For instance, if the trajectory is supposed to reach a certain value in a certain point and EC is not able to reach it, it is probable that this point error will sum up to the error of the next evolved point as a constant in the fitness calculation. An example of the execution of an evolved action can be seen in Figure 7.16.

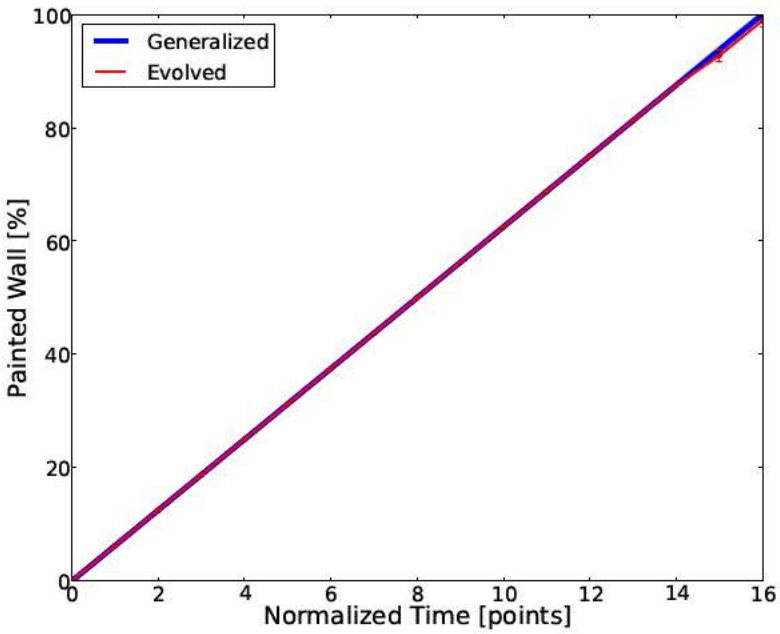

Figure 7.15: *Unidimensional temporal plot of the generalized action, and the trajectory obtained by executing the EC winner joint parameters. As seen for a linear painting task (one wall square is painted in each step), the performance is very accurate.*

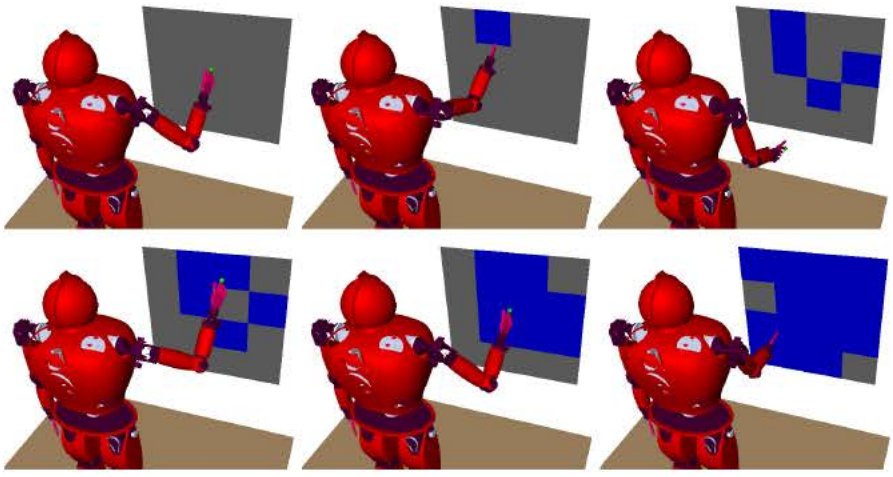

Figure 7.16: *Action execution for a PAINT command. Each square changes its color when the hand gets closer than a specific distance.*

The criteria of success or fail for a given goal-directed imitation, strongly depends on the action being imitated. In the case of the CLEAN task, we set an acceptable error for each feature point. However, in the case of the 'painting' task, the criteria is related with the total number of squares painted.

7.4 Guided Motor Primitives

Two experiments have been performed in this section. Both experiments are performed in a simulated environment with a model of the humanoid robot TEO. The library of primitives used is extracted from the random trajectory shown in Figure 6.5. The first experiment involves only object spatial features, and its objective is to generate a robot joint trajectory which leads to a feature trajectory equal to the objective one. This feature trajectory is similar to a cleaning movement, and it is encoded as a CGDA where the object tracked is in one of the robot's hands. The second experiment consists in having the robot paint a wall. The painting process is encoded as a CGDA, but this time the object tracked is the wall.

7.4.1 Executing a Spatial Task: Cleaning

The goal is to reproduce a generalized feature trajectory extracted from a set of human action demonstration repetitions. This generalized feature trajectory contains only spatial features (X,Y,Z). The demonstration repetitions which leaded to the generalized action were recorded using a real Kinect device tracking a colored marker. Seven repetitions of the demonstrated action were recorded and used to generate the generalized action. The action can be described as: keeping object orientation fixed, move it over the perimeter of a circle of 30 cm of diameter for one revolution. For simplicity of explanation, let us name this action as CLEAN.

For this experiment several values of ξ were used to generate different sets of primitives. As a remainder, ξ is the threshold of similarity between two primitives to consider them similar or not. With these primitives, SICS was applied to find a primitive combination which obtains an error lower than $\varepsilon = 20$ for each point of the generalized feature trajectory. This ε represents a 20 mm error in this experiment because the feature space was set to millimeters. The resulting feature trajectories for the cleaning task can be seen in Figure 7.17.

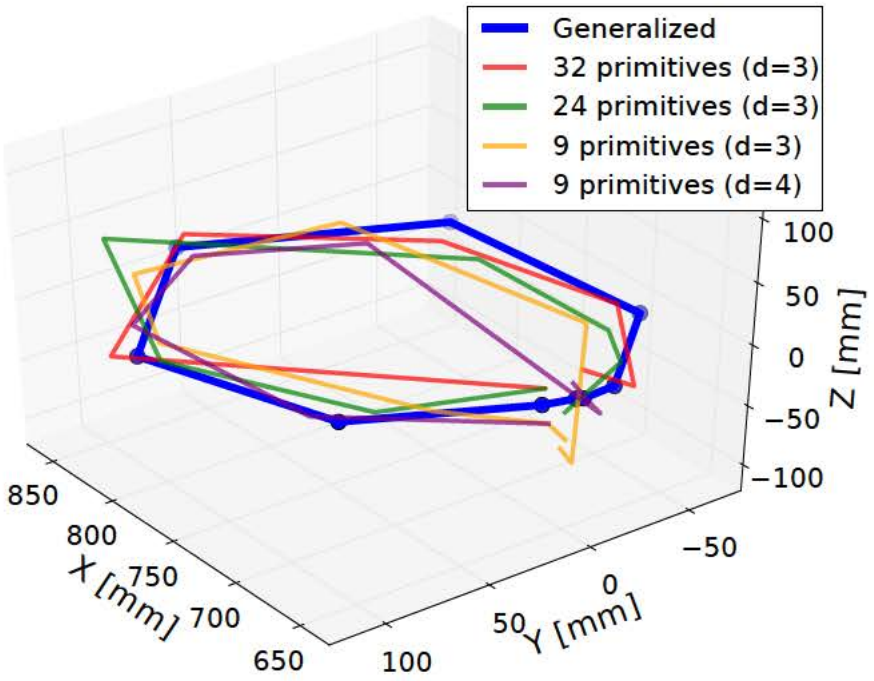

Figure 7.17: *Cleaning: Plot superposing the generalized feature trajectory (blue) and several feature trajectories resulting from executing the sequence of primitives selected by SICS. Results are dependent on the number of primitives available to choose from (given by the threshold of similarity ξ) and the maximum tree depth d. The primitives used belong to the same set of segments, but in function of ξ, the most diverse are selected.*

In general, allowing a greater maximum tree depth d increments the possibilities of finding a solution, but results in an exponential evaluation complexity. The use of a larger set of primitives leads to a greater number of possible combinations, so the algorithm is more likely to find a solution.

7.4.2 Executing a Non-Spatial Task: Painting

In the second experiment the task goal is to paint a wall, without teaching the robot the necessary motor parameters. In the simulated scenario, there is a wall, composed by 16 small gray squares, in front of the robot. The action goal is to change the color of all the squares. The generalized action has been created synthetically and it has only one feature (plus time). This feature represents the percentage of wall that has been painted, which increases constantly with time. The generalized feature trajectory is

split in 16 points.

The generalized feature trajectory is used as the reference and SICS is used to find primitive sequences. Every time the robot's hand gets closer than 120 mm to a wall square, this square changes its color. Cost is evaluated when a primitive sequence is executed, by analyzing wall features (counting the number of colored squares).

Several values of ξ were used to generate different sets of primitives. With these primitives, SICS was applied to find a primitive combination with an error lower than $\varepsilon = 0.1$ for each point. This ε represents a 0.1% error in this experiment as the feature space was set to represent the percentage of the painted wall. The resulting feature trajectories for the painting task can be seen in Figure 7.18.

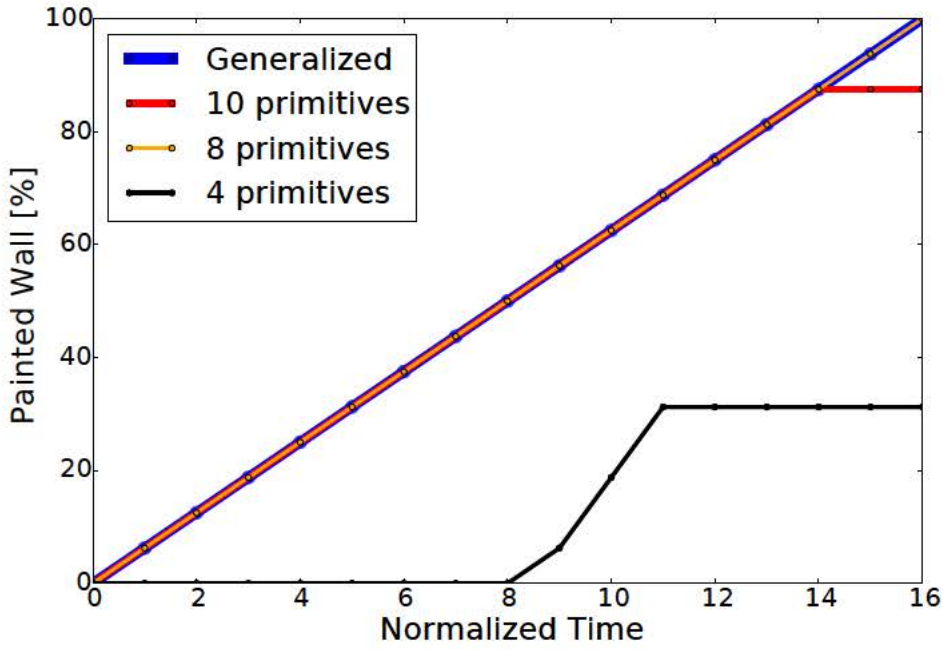

Figure 7.18: *Painting: Plot superposing the generalized feature trajectory (blue) and several feature trajectories resulting from executing the sequence of primitives selected by SICS. Results are dependent on the number of primitives available to choose from (given by the threshold ξ). Remember the plot is not representing joint or spatial trajectories, but feature ones.*

The results show that the system can accomplish the painting task with a small number of primitives (example in Figure 7.19).

By increasing the number of primitives, the probabilities of a correct performance rise. However, a new primitive may alter a path and lead to worse results for posterior

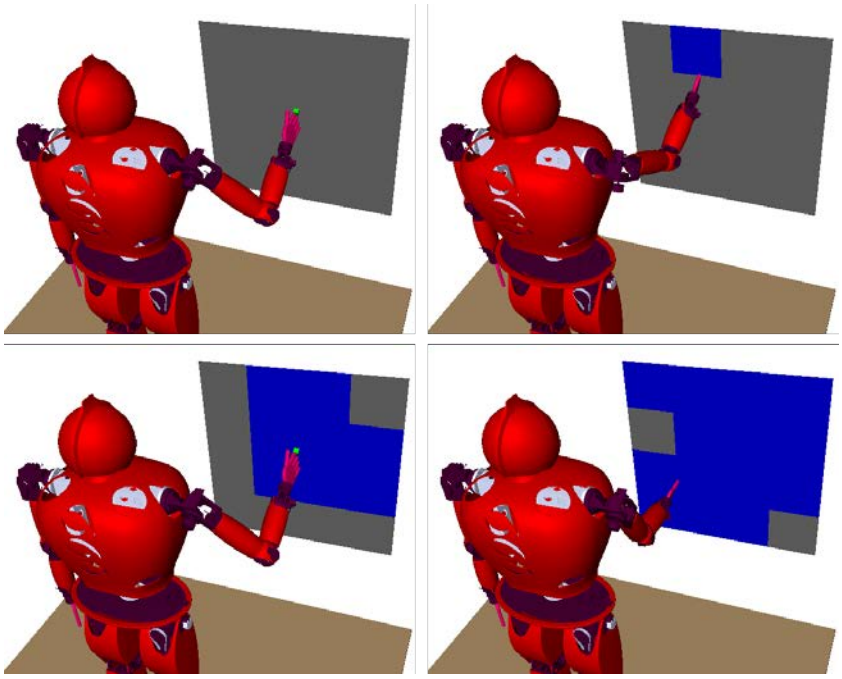

Figure 7.19: *Several screenshots taken during the painting experiment. Motor sequence obtained from combining primitives using SICS.*

points. This is the case of the figure when using 10 primitives.

7.5 Friction and Gravity Compensation

The experiments have been performed using the arm of the real humanoid robot TEO. A single 1 DoF robot joint was tested, in order to avoid the high-dimensionality and coupling effects of many DoF (similarly to (Mallon, van de Wouw, Putra, & Nijmeijer, 2006)).

The humanoid robot joint used was the robot's left shoulder, which is moved by a Maxon brushless EC flat motor. It has a torque constant of 0.0706 Nm/A. The motor driver has an internal current loop with a PI regulator, with constant $K_p = 0$ and $K_i = 0.1651$. The gearbox is a Harmonic Drive CSD-25 with a reduction factor of 160. Joint position is measured using an optical relative encoder attached to the motor. Velocity is obtained by numerical differentiation of the position signal.

7.5.1 Computation

The robot arm weight m is 4.446 kg (including hand and electronics), and it has a length L of 0.82 m. The gravity compensation term of the control was computed as the torque caused by the arm modeled as a punctual mass at its center of gravity. Considering h as the height of the center of gravity with respect to its lowest position, being a single joint, this term is trivial to be calculated. Assuming q_1 as the angle between the arm and the trunk, the potential energy of a mass situated at $L/2$ from the shoulder is:

$$U = mg_0\, h = mg_0(L/2)(1 - cos(q_1)) \tag{7.1}$$

Then, the gravity torque term is:

$$g(q_1) = \frac{\partial U}{\partial q_1} = mg_0(L/2)\sin(q_1) \tag{7.2}$$

The friction compensation term was determined by the procedure derived from (Virgala & Kelemen, 2013), which leads to a model like the one in Equation (6.15). When high torques are applied to the motor, leading to high velocities, the motor is not able to stabilize its velocity before reaching the mechanical limit. This results in a steeper slope on the posterior regression. This effect can be seen in the shortest curves in Figure 7.20. Fourteen different constant motor torques were tested, including both movements against and in favor of gravity, ranging between -0.0706 Nm and 0.0706 Nm (ranging from -1 A to 1 A). The robot's arm is placed so that positive motor velocities go against gravity, and negative velocities are in favor of gravity. To determine the friction term, a summary of the process can be seen first in Figure 7.20, where the stability velocities are measured, and then in Figure 7.21, which depicts the performed linear regressions.

The linear regressions obtained without manual tunning resulted in:

$$\tau_f(\dot{q_1}) = \begin{cases} 0.009\,\dot{q_1} + 0.490 & : \ \Delta U > 0 \\ 0.005\,\dot{q_1} - 0.586 & : \ \Delta U < 0 \end{cases} \tag{7.3}$$

A manual adjustment of these linear regression parameters resulted in:

$$\tau_f(\dot{q_1}) = \begin{cases} 0.006\,\dot{q_1} + 0.4 & : \ \Delta U > 0 \\ 0.001\,\dot{q_1} - 0.7 & : \ \Delta U < 0 \end{cases} \tag{7.4}$$

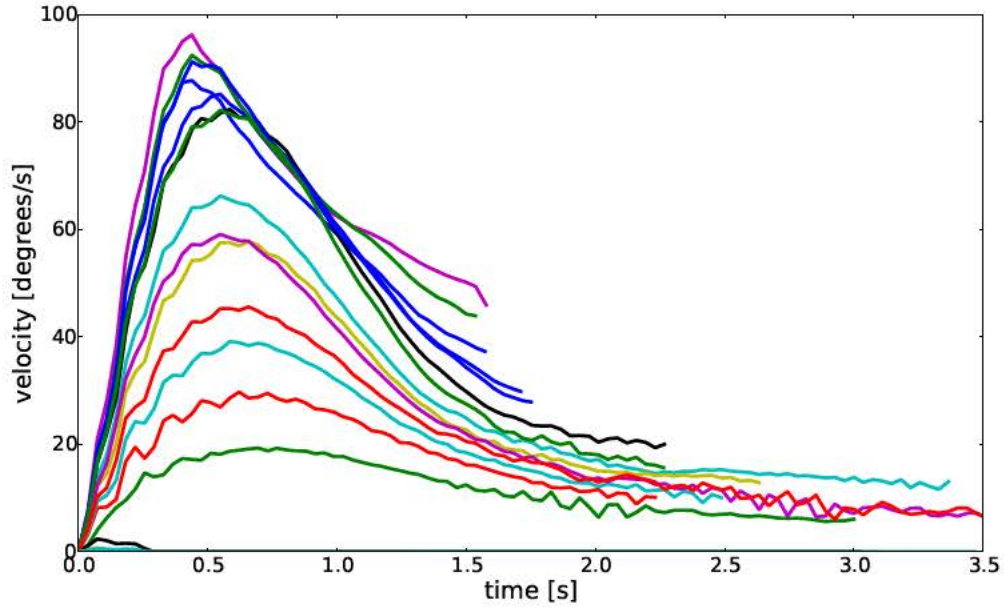

Figure 7.20: *Velocity vs. time curves, with constant torques applied in each curve. There is a proportional linear dependence between current in the motor and torque applied.*

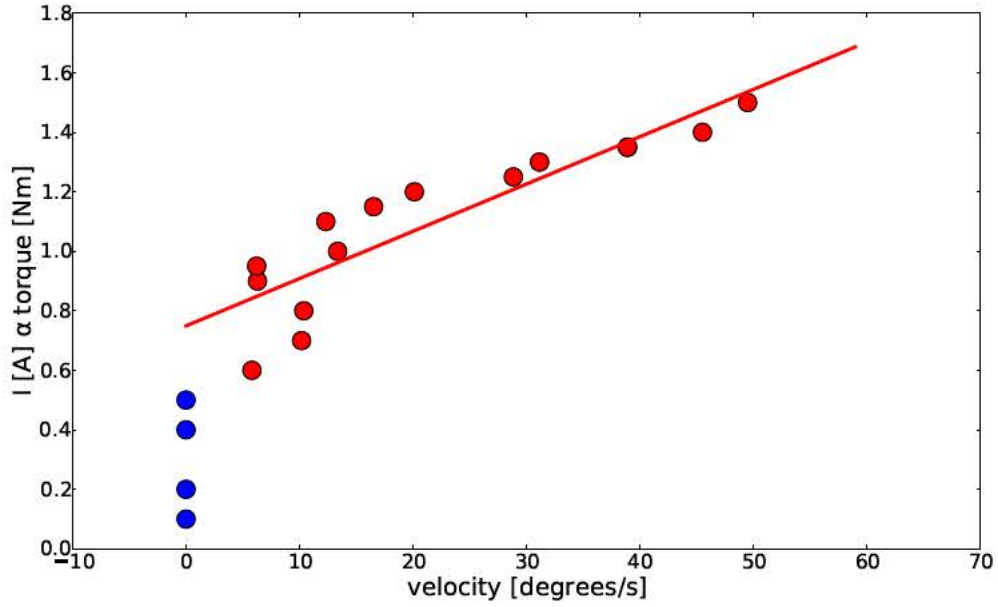

Figure 7.21: *Stabilization velocity vs. the torque applied. The Coulomb friction (blue) and the viscous friction (red) can be seen. The same procedure is applied to negative velocities.*

To adjust to a symmetric joint friction model, either of the equations may be selected to fix the parameters of the model. Here, we have used the parameters of $\Delta U < 0$ case of (7.4), resulting in the joint friction term τ_{fj} of the controller:

$$\tau_{fj}(\dot{q_1}) = \begin{cases} 0.001\,\dot{q_1} + 0.7 & : \ \Delta U > 0 \\ 0.001\,\dot{q_1} - 0.7 & : \ \Delta U < 0 \end{cases} \tag{7.5}$$

All other frictions will be considered part of the gravity friction term τ_{fg}. The position-dependent parameter of τ_{fg} was experimentally adjusted to $0.0025\,q_1$. The final expression of τ_{fg} is computed as (7.4) minus (7.5) plus the position-dependent parameter τ_{fg}.

$$\tau_{fg}(q_1, \dot{q_1}) = 0.005\,\dot{q_1} + 0.0025\,q_1 - 0.3 \tag{7.6}$$

7.5.2 Evaluation

The compensators were evaluated activating the ZFZG controller. A well designed Zero-Friction Zero-Gravity controller would maintain constant, or tightly bounded, velocities in absence of external perturbations (beyond the one initiating the movement). To test whether these conditions are applicable to our system, several interactions with the arm were performed. A single push was given to the arm, letting it move freely while recording its velocities. This experiment was repeated while pushing the robot arm with different forces. Several velocity profiles for different pushes can be seen in Figure 7.22.

Curves reaching higher peak values in the figure represent larger forces exerted by the human. The peak of each curve roughly represents the instant where the robot arm is let free. An example of one of these interactions using the ZFZG controller can be seen in Figure 7.23.

Results from Figure 7.22 showed that the combination of friction compensation and gravity compensation was successful at maintaining bounded velocities in absence of external perturbations for velocities below 30-35 degrees/s. Results also show that the friction model is not adequate for velocities above 30-35 degrees/s. For instance, the light blue and purple curves do not maintain their values after their peaks. This could be explained because of the unmodeled non-linearities of the friction function at these velocities. A video of the implementation was shown in the IEEE Humanoids 2014

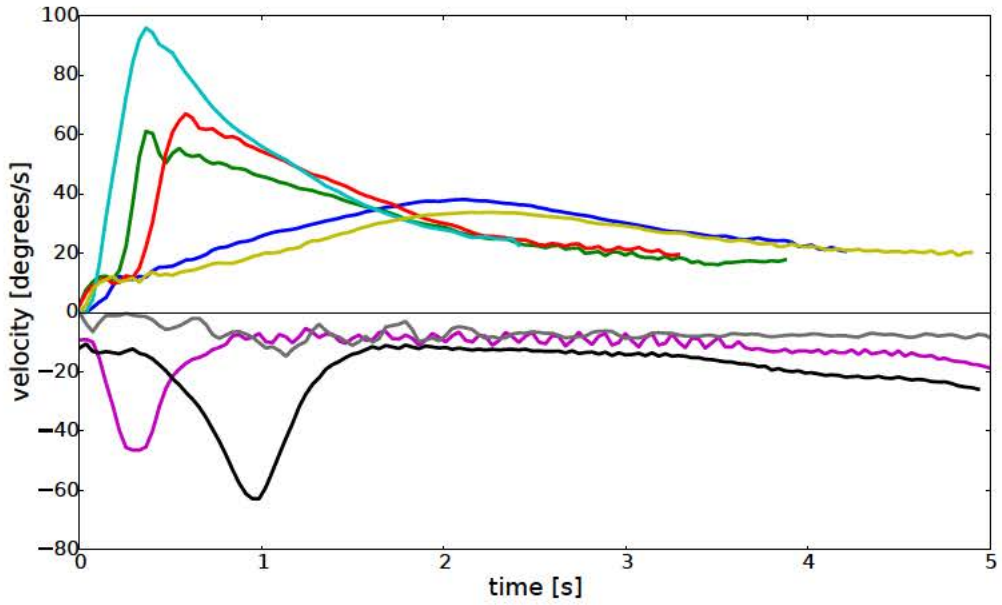

Figure 7.22: *Velocity profiles for several initial 'pushes' with the Zero-Friction Zero-Gravity controller (ZFZG).*

Figure 7.23: *Sequence of the movement of the robot arm using a Zero-Friction Zero-Gravity controller. When the user pushes the arm, it moves freely in the direction of the applied force.*

conference (Morante, Victores, Martinez, & Balaguer, 2014), and can be seen online[1].

Our linear friction model may look simplistic, as each part in the mechanism includes its own non-linearities. However, it accounts for physical components (such as gearboxes and joint limits) that are found in robotic systems.

It is important to recall that it was not an objective to create a highly detail model of friction and gravity. We just aimed at aiding in moving the robot, to be useful in paradigms like Programming by Demonstration. In this target scenario, guidance movements are performed at low velocities. This fact makes our model useful, despite not being adequate for high velocities scenarios.

[1] http://dai.ly/x2vjrfs

Chapter 8

Discussion and Conclussions

In this chapter, a discussion about each development introduced in this thesis is presented. Conclusions and future works are also depicted at the end of the chapter.

8.1 Normalization on Demonstration and Feature Selection

In our demonstration and feature selection experiment, all the features were measured in the same distance units. This may not be case for other experiments, so data normalization may be a requirement. While it may be necessary, **normalization may dangerously distort the data**. The following is a description derived from of our own experience upon using the different methods presented in Subsection 4.2.1.

- MinMax: When normalizing each feature within its empirical limits, the noise in the signal may be amplified. For instance, a flat signal having sensor noise **may result in a signal where the noise has been greatly amplified**.

- Standardized: **Outliers may attenuate the signal**. Imagine a sine wave signal with a far outlier (e.g. a sensor failure). Having outliers often produces a significant effect on the mean and standard deviation. When the signal is subtracted the mean and divided by the standard deviation, it may be completely flattened.

- Whole Experiment Normalization: **The data may be distorted by both of the problems already explained**.

- Physical Limits: This normalization implies that **the designer has a previous knowledge of each feature limit**. In some cases, the limits will be imposed be the sensor, while in other cases (e.g. derived features and combinations), mathematical limits must be determined. There is an additional issue when tasks do not expand over the whole space because they are instead concentrated in a small region, and features limits have been set with large ranges. When the signal is normalized, it may be flattened, becoming no more relevant than noise.

It is an open discussion what technique is the most appropriate, and the authors believe it will depend on the learning context.

8.2 Limitations of CGDA

We have used both Dynamic Time Warping and the Euclidean distance for recognition. Regarding the Euclidean distance results, there is a clear improvement in the error between the trajectories, when comparing to DTW. However, a numerical comparison would not be fair, e.g. comparing DTW to the distance error would be assuming that the Euclidean distance is the "correct" metric. On its influence on execution, this change forces the system to accomplish the feature variations in the correct order.

We have introduced an evolution strategy called Incrementally Evolved Trajectories (IET). This method has allowed us improve the performance of execution for non-spatial features. IET shows an incremental time of computation for a sequence of points until the valid space becomes very restricted. In these situations, we have observed peaks in the computation time. Despite this fact, it allows experiments that were previously not feasible. The motor performance, in terms of velocities and acceleration, is not taken into account. This thesis' focus was on developing a feasible goal-directed framework. **Adapting this framework to work with real robots will imply a refinement of the commanded motor signals, as well as determining the optimal number of required human demonstrations**.

Regarding the results obtained in the experimental section, we have demonstrated that our CGDA approach is useful for execution even for non-spatial features. Thanks to the use of the Euclidean distance, the trajectories can be evolved obtaining a high degree of similarity, and this error is configurable for each evolution. Further analysis need to be performed in order to assure that this method is applicable to other non-

spatial features. **The biggest limitation of CGDA, is that execution, at this moment, can only be performed in simulation**. The techniques developed are not applicable to the real world, as it is not feasible to perform thousands of iterations with a real robot.

8.3 The Benefits of Human-Robot Interaction

The inclusion of a human in the process of creation of Guided Motor Primitives is very valuable. By benefiting from human-robot interaction, the complexity of the framework is reduced when generating primitives.

Other self-discovery approaches, such as motor babbling, require programming joint-limits and computing self-collisions. Despite GMP and motor babbling aim to fill similar technological niches, our approach simplifies the process as it is the human who leads the exploration of the joint space.

A limitation in our approach is that it is limited to simulation. As primitives are combined and reproduced, collisions must be checked. This limits the applicability of this approach to a real robot. A solution to this problem is to find the correct path in simulation, and then transfer it to a real robot.

8.4 A More Accurate Compensation

The experiments of this set show, in general, an acceptable performance of the ZFZG controller tested. More accurate friction models and identification procedures could lead to improved controller behaviors under high joint velocity conditions, and would also aid in maintaining the stability of low velocities. We also consider that using different dynamic robot models (pendulum-like models, or even the complete dynamic model) may improve the results.

With the potential increase in complexity of the complete humanoid robot model, it may be unavoidable to use machine learning algorithms which lighten the efforts necessary to obtain a reliable friction and gravity compensation. They could also capture the non-linearities present in the system.

These improvements may help at improving the general performance of the controller. However for the use-case considered in this thesis, the controllers developed are adequate, due to the low velocities of the interaction.

8.5 Conclusions

The main contribution of this thesis is enabling a robot to perform tasks without previous knowledge of the kinematic parameters to perform them. The Continuous Goal-Directed Actions (CGDA) analysis allows us to focus on task accomplishment, rather than solely on motor imitation. By encoding objects' features, the model extracted from the demonstrations represents a goal model, which is encoded without kinematic parameters. This approach is novel within the robot imitation paradigm. Let us list the main contributions per studied topic:

1. A procedure has been develop to encode actions by their effects. A generalized model based on goals can be extracted from a set of demonstrations. Using the CGDA encoding, robots can understand the goals of actions. CGDA has provided the tools to analyze actions as goals to achieve.

2. The method based on Dynamic Time Warping has proven useful in distinguishing and comparing different tasks based on object features. Comparing continuous tasks' goals is novel, and allows to perform a similarity analysis of tasks.

3. Including demonstration and feature selection in the process has been very useful. The results are impressive and save us time for performing experiments. Despite feature selection is a well known problem, demonstration selection had not been analyzed previously in the literature.

4. With CGDA, we are enabling a robot-configuration independent task accomplishment, also avoiding the correspondence problem. From the set of evolutionary strategies presented, Incrementally Evolved Trajectories allows us to execute a feature trajectory with temporal dependencies, such as painting a wall. It reaches an acceptable degree of performance in execution of goal-directed tasks. Regarding Guided Motor Primitives, after only 47 seconds of human interaction, the robot has learned enough primitives to perform several tasks. By combining the discovered primitives with the Sequential Incremental Combinatorial Search the robot is able to perform the execution of goal-directed tasks, proving also that GMP are reusable for different tasks. The results show an increase in performance when using a greater number of primitives, as it has more combinations from

which to choose. Furthermore, longer combinations of primitives also tend to improve the results.

5. If robots are expected to work alongside humans, it is reasonable to believe that friction and gravity compensation will become a future standard for manufacturing robots. Our friction compensation method is acceptable for low velocity's tasks. While it is useful to drive a single robot arm joint, scalability needs to be further analyzed. However, a simplification of the friction model is feasible to be implemented in a full robot.

8.6 Future Lines of Research

A number of open challenges can be presented:

- One future line of research could be the integration of the presented Continuous Goal-Directed Actions framework with the Programming by Demonstration one. Feature selection could be fed with demonstrator's features as used in classical Programming by Demonstration experiments. A successful integration of CGDA and PbD could require having the feature selection process decide which features (object or human ones) are most relevant for recognizing or executing an action. Feature selection could also be fed with features beyond spatial and kinematic parameters.
- An alternative algorithm for the filtering block in feature selection could be found. Z-score has proven useful, but its assumption of having enough relevant features may not be acceptable for all the learning scenarios.
- Friction compensation for multiple joints will involve non-linear couplings that may require machine learning algorithms to be modeled.
- Execution of CGDA actions in real robots remains the biggest open challenge of this thesis. A possible path to achieve it could be to develop algorithms to reduce the number of iterations required to be able to perform a task. Reinforcement learning algorithms may become useful in solving this challenge.
- Another work to be developed based on this thesis could be parametric actions. Actions which depend on additional information to be correctly performed are

not taken into account at the current state of development. This is the case, for instance, of the action *paint*: it is not the same to *paint blue* than to *paint red*. In our feature space, these actions would end in different coordinates for the *color* dimension. The solution to this problem could be to mix the structure here presented with semantic information, in a similar way to our previous work with objects (Victores, Morante, Jardón, & Balaguer, 2013b).

With the advancements presented in this thesis, we are pioneering a new area of research of robotics. Programming by Demonstration is a useful framework for current robots, but future robots will evolve to satisfy new forms of interaction. Today we must demonstrate each task to a robot, for it to mechanically repeat it. In the future, robots should be able to reuse and combine previous experiences to perform new, or even previously unseen, tasks. The author looks forward to seeing how the research presented in this thesis may aid in this direction.

Appendices

Appendix A

Opinion: Cryptobotics: Why robots need cyber safety

With the expected introduction of robots into our daily lives, providing mechanisms to avoid undesired attacks and exploits in robot communication software is becoming increasingly required. Just as during the beginnings of the computer age (Pfleeger & Pfleeger, 2002), robotics is established in a "happy naivety", where security rules against external attacks are not adopted, assuming that robotics knowledgeable people are well intended. While this may have been true in the past, the mass adoption of robots will increase the possibilities of attacks. This fact is specially relevant in defense, medical and other critical fields involving humans, where tampering can result in serious bodily harm and/or privacy invasions. For these reasons, we consider that researchers and industry should deploy efforts in cyber safety and acquire good practices when developing and distributing robot software. We propose the term *Cryptobotics* as a unifying term for research and applications of computer and microcontrollers security measures in robotics.

A.1 Stating The Problem

The problems that the field of robotics will face are similar to those the computer revolution faced with the widespread of the Internet 30 years ago. Among the common attacks computers may suffer, there are: denial-of-service, eavesdropping, spoofing,

tampering, privilege escalation or information disclosure for instance. To these problems, robots add the additional factor of physical interaction. While taking the control of a desktop computer or a server may result in loss of information (with its associated costs), taking the control of a robot may endanger whatever or whoever is near.

As robots become more integrated on the communications networks, it seems appropriate to reuse the tools designed for web applications in order to controls the robots. However, the authors consider there are differences between regular computers communicating through the network, and robots performing the same actions. Mohanarajah (Mohanarajah, Hunziker, D'Andrea, & Waibel, 2015) states differences between web and robotic applications: "Web applications are typically stateless, single processes that use a request-response model to talk to the client. Meanwhile, robotic applications are stateful, multiprocessed, and require a bidirectional communication with the client. These fundamental differences may lead to different tradeoffs and design choices and may ultimately result in different software solutions for web and robotics applications". To these differences, we could also add the real-time constraints that characterize robotics applications. Despite other sources of issues, like software bugs or vulnerabilities (buffer overflow, command injection, etc. (Tanenbaum, 2009)), we consider that communications currently are one of the main vulnerabilities in robotics.

A number of fields in robotics where security and privacy are particularly relevant can be addressed.

- Defense: The military field should be very aware of the best practices in cyber security to be followed regarding its robots. Unmanned aerial vehicles, commonly called "drones", are being destined to surveillance and also to combat missions. Common sense dictates that any communications with these vehicles should be encrypted (Javaid, Sun, Devabhaktuni, & Alam, 2012), but reality shows us differently. For example, in the year 2012 it was reported that only between 30 and 50 percent of America's Predators and Reapers (two of the most used drones in US) were using fully encrypted transmissions[1].

  Situation: a non-authorized entity eavesdrops surveillance images of drones, takes its control, exploiting a non-encrypted connection, and crashes it into a populated area.

[1]Most U.S. Drones Openly Broadcast Secret Video Feeds: http://www.wired.com/2012/10/hack-proof-drone/

- Telerobotics in Space: Developments are being carried out to develop telerobotics infrastructures (e.g. Robonaut, METERON, etc.). The Mars rover is another example of telerobotics in space. NASA is very concerned about information security, as can be understood from the testimony given in 2012 by Paul K. Martin, NASA's Inspector General (Martin & General, 2012). Martin highlights some issues within the agency's structure, such as a lack of awareness of IT security, slow pace of encryption for NASA devices, and the ability to combat sophisticated cyber attacks. Due to the cost of the projects carried out by space agencies, it is critical to avoid any undesired interferences by non-authorized agents in the experiments.

  Situation: a non-authorized entity takes control of a robot inside International Space Station and sabotages an ongoing experiment.

- Telemedicine and Remote surgery: This exciting field can make remote surgery become an everyday reality, where experts can operate patients from the other side of the world. While this is beneficial to society, we must consider the potential dangers. In 2009, the Interoperable Telesurgery Protocol (ITP) (King et al., 2009) was proposed as a preliminary specification for interoperability among robotic telesurgery systems. Recently, the fact that ITP does not use any form of encryption or authentication was discovered[2]. This is an obvious system exposure to exploits using a man-in-the-middle attack for taking control of the robot (Bonaci et al., 2015).

  Situation: a non-authorized entity takes control of a surgery robot during an operation, endangering the life of the patient.

- Household robots: This market is growing both in research and commercially available robots. Robots will be used as assistants at home. For instance, one of these projects is Care-O-bot (Hans, Graf, & Schraft, 2002), a robotic assistant in homes. In one of the available versions, this robot is equipped with microphones, cameras and 3D sensors. This set of sensors can collect a huge amount of information, which must be protected (Denning, Matuszek, Koscher, Smith, & Kohno, 2009). Service robots may one day also collect data about the health status of a person; law regulations require that this data is handled with extra care.

[2]Interoperable Telesurgery Protocol (ITP) Plaintext Unauthenticated MitM Hijacking: http://osvdb.org/121842

Situation: a non-authorized entity takes control of a household robot and obtains streams of images with private data.

- Disaster robots: Since the Fukushima Daiichi nuclear disaster in 2011, the robotics community has increased its efforts to build and deploy robots for disaster scenarios. One of the expected tasks these robots will have to face in a disaster scenario is related to accessing and repairing/disconnecting dangerous systems. Due to the potential danger that may arise in these situations (Vuong, Filippoupolitis, Loukas, & Gan, 2014), robots should not be able to be externally modified by an external attack.

  Situation: a non-authorized entity takes control of a robot deployed to disconnect a nuclear platform that may suffer a partial meltdown, and can thwart the disconnection operation.

A.2 Current State of Security in Mainstream Robotic Software

Robots are a combination of mechanical structures, sensors, actuators, and computer software that manages and controls these devices. Mainstream practices in robotics involve component-based software engineering. Each component is designed as an individual computer program (e.g. a motor moving program) which communicates with other components using predefined protocols. While a large quantity of software libraries for communication already exist, the robotics community has developed a number of "software architectures". Currently, one of the most popular robotics-oriented architecture is ROS (Robot Operating System) (Quigley et al., 2009). Another co-existing architecture is YARP (Yet Another Robot Platform) (Metta et al., 2006). Both systems work similarly: A system built using ROS or YARP consists of a number of programs (nodes or modules), potentially on several different hosts, connected in a peer-to-peer topology.

According to ROS documentation[3]: "Topics are named buses over which nodes exchange messages. Topics have anonymous publish/subscribe semantics (...) In general, nodes are not aware of who they are communicating with". From the point of view of security, this anonymous communication scheme is a welcome sign towards exploits (McClean, Stull, Farrar, & Mascareñas, 2013). Messages are sent unencrypted through TCP/IP or UDP/IP. The default check performed is an initial MD5 sum of the message

[3] http://wiki.ros.org/Topics

structure, a mechanism used to assure the parties agree on the layout of the message. Some researchers have developed an authentication mechanism for achieving secure authentication for remote, non-native clients in ROS (Toris, Shue, & Chernova, 2014). While it can increase the security of the overall system, without data encryption, an eavesdropper could acquire non-encrypted information.

Part of the ROS community is dedicating efforts to integrating OMG's DDS (Data Distribution Service) as a transport layer for ROS 2.0 [4]. A preliminary alpha version has just been released. DDS is a standard specification followed by several vendors, for a middleware providing publish-subscribe communications for real-time and embedded systems. RTI provides plugins which comply with the DDS Security specification including authentication, access control and cryptography. It would be a big step forward for securing our robots if ROS 2.0 aimed to comply with the DDS Security specification as well.

YARP states among its documentation[5]: "A [default] new connection to a YARP port is established via handshaking on a TCP port. So everyone who can access this TCP port can connect to your YARP port. So if you are not behind a firewall, you are exposing your YARP infrastructure to the world (...) And if your application is vulnerable to corrupted data, it is a security leak." Other YARP documentation reads clearly[6]: "If you expose machines running YARP to the Internet, expect your robot to one day be commanded to make a crude gesture at your funders by a script kiddie in New Zealand". However, an authentication mechanism can be activated in YARP, which adds a key exchange to the initial handshaking in order to authenticate any connection request. It has been enabled by default so it is always compiled. However to preserve backward compatibility the feature is skipped at runtime if the user does not configure it by providing a file that contains the authentication key.

Additionally, a new port monitoring and arbitration (Ali Paikan & Natale, 2014) functionality inside YARP has been used to implement a LUA encoder/decoder of data[7]. Data is passed through a Base64 encoder before being sent, and decoded upon reception at the target port. A similar mechanism could potentially be used to encrypt and decrypt the data.

[4] http://design.ros2.org
[5] http://wiki.icub.org/yarpdoc/yarp_port_auth.html
[6] http://wiki.icub.org/yarpdoc/what_is_yarp.html
[7] http://wiki.icub.org/yarpdoc/coder_decoder.html

A limited amount of other works have also focused on securing robot communications. In (Groza & Dragomir, 2008) they implement an authentication protocol to assure the authenticity of the information when controlling a robot via TCP/IP. However, they do not implement encrypted communications. In (Coble, Wang, Chu, & Li, 2010) they implemented a hardware system that verifies integrity and health of the system software (to avoid tampering) in telesurgical robots. Regarding the previously mention ITP protocol, some researchers are working on security enhancements (Lee & Thuraisingham, 2012). One commercially available robot that does take cyber security into account is BeamPro, a telepresence robot [8] where secure protocols, symmetric encryption, and data authentication are used, thus providing security and privacy.

Secure communications are even more important in new trends in robotics which aim at outsourcing computation, namely *Cloud Robotics*. In this paradigm, robots use their sensors to collect data, and then upload the information to a remote computation center, where the information is processed, and may be shared with other robots. Rapyuta (Mohanarajah et al., 2015) is an example of this paradigm where the tecnologies used (e.g. WebSockets) allow to secure the information.

Another usual way of communications between robot's devices is through communication buses (CAN, EtherCAT, etc.). Currently, none of the traditional field buses offers security features against intentional attacks (Dzung, Naedele, Von Hoff, & Crevatin, 2005). However, those based on ethernet could potentially make use of the security measures included in TCP/UDP/IP. For instance, secure routers (e.g. EDR-G903), include firewalls and VPNs, and support EtherCAT.

A.3 Guidelines and Security Checks

By highlighting current cyber security issues and creating the term *Cryptobotics*, we hope to foster research in cryptography and robotics. As an initial step for new users of cyber security, we may outline some basic security measures to be taken by any robotics project.

1. Any wireless connection between the robot and the operator, or between robots, should be encrypted.

[8] https://www.suitabletech.com/

2. Always assume the environment is hostile when establishing communications. Be sure that eavesdroppers cannot perform a man-in-the-middle attack by establishing key negotiation for insecure channels.

3. Any communication module (either a robot, a robot part, an operator, etc.) should have a unique identifier, and each transmission should be able to be verified at any moment to confirm the parties' identities.

4. Any device is password protected, and its content encrypted.

5. When a high risk of message interception exists, communications should be implemented following a perfect forward secrecy framework, where each communications is encrypted with a different key, assuring no previous (or future) communication can be revealed if one key is discovered.

6. Access logs exist and are updated with each transmission. This may not prevent an attack, but helps track where an attack came from. These files should be encrypted and should not be accessible with standard user privileges. A separate password must protect them.

7. A firewall shielding access to internal communications should exist.

8. A place where passwords, keys and a registry of allowed users are stored should exist. This place is the most critical device, and must be protected adequately.

An optional but strongly recommended requirement is to use audit and reviewed open-source encryption software. Many public and well tested algorithms to encrypt communication exist. Whenever possible, use largely accepted and community-checked algorithms.

In terms of selection among algorithms, there are mainly three components in a cyber security protocol: a **public-key** cryptosystem, used to derive **symmetric-keys** (also called private-keys) which ensure confidentiality of data (Gupta, Gupta, Chang, & Stebila, 2002), and a **message authentication code** (MAC) used to authenticate the data. While this selection is a design decision open to system architects, the SSL protocol recommends a *cipher suite* (combinations of algorithms) composed by RSA-RC4-SHA: RSA as the public key, RC4 as the symmetric key, and SHA as a hash MAC.

A.4 Discussion

A big market of opportunities for research regarding cyber safety in robotics exists. Most robots are not yet prepared, from a security point of view, to be deployed in daily life. The software is not prepared to protect against attacks, because communications are usually unencrypted.

Regarding the dates of the exploits presented, and the current hype in deployment of daily robotics (vacuum cleaners, amateur drones, etc.), *Cryptobotics*, understood as a mix of cyber security and robotics, comes just in time to prepare these systems to be safely used.

An important issue to be discussed is whether the implementation of encrypted communications may affect the performance, specially in real-time systems. The question about performance is highly dependant on the hardware, the software and the network used. Encrypted communications on the Internet (https, ssh) show us that it is possible to perform secure communications and offer remote services. For instance, Adam Langley (Google Senior Software Engineer) has stated: "when Google changed Gmail from http to https (...) we had to deploy no additional machines and no special hardware. On our production front-end machines, SSL/TLS accounts for less than 1 of the CPU load."[9]. From our experience in humanoid robotics, a 1% overhead (while respecting determinism in time) can be acceptable if it means our devices can be less vulnerable to cyber attacks. Could an 8 MHz microcontroller perform real-time encryption? Is it reasonable to implement authentication mechanisms along field buses in time-constrained scenarios? This article intends to raise awareness for developers to determine whether it is viable to integrate these mechanisms depending on each specific use case.

Some people may ask why these problems have not been addressed previously. In recent years, intrinsically safe industrial robots, the rise of domestic robots, and the use of mobile robots in public spaces, have arisen issues that the robotics community did not have to face in its previous 60 years of existence. Researchers are now focused on developing applications to make robots useful, which may have made cyber safety a low priority.

[9] https://www.imperialviolet.org/2010/06/25/overclocking-ssl.html

Appendix B

TEO, The Humanoid Robot

In this appendix, we provide technical information about the humanoid robot TEO. Simulated models have been constructed following these data.

- Axes and directions of rotation are shown in Figure B.1.
- Denavit-Hartenberg directions are shown in Figure B.2.
- Denavit-Hartenberg parameters are shown in Figure B.3.
- CAN (Controller Area Network) identification number of the joints are shown in Figure B.4.
- YARP identification number of the joints are shown in Figure B.5.
- Mechanical links lengths distribution are shown in Figure B.6.
- Mechanical links lengths measurements are shown in Figure B.7.
- Position of the Center of Masses in the concentrated mass model are shown in Figure B.8.

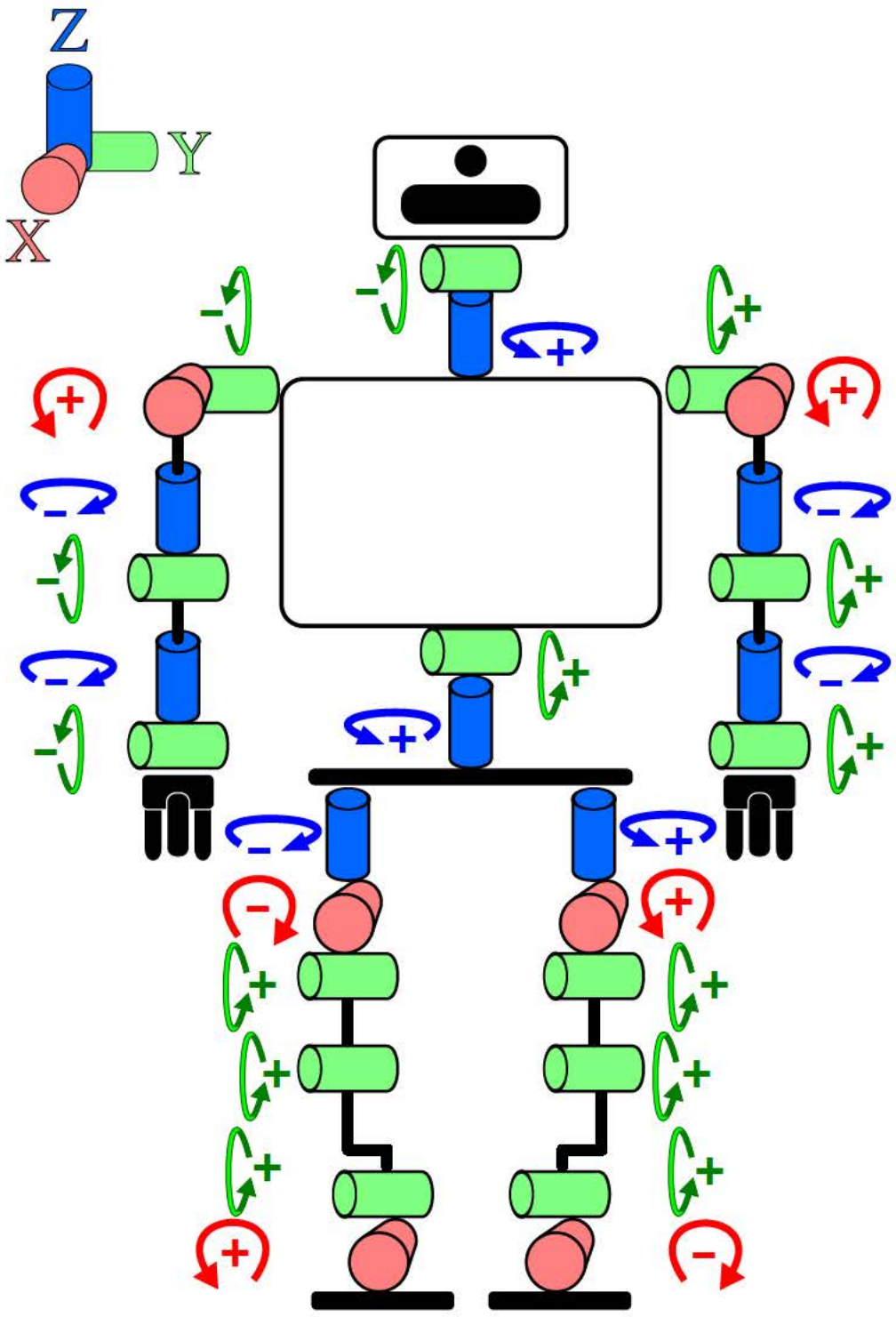

Figure B.1: *Axes and directions of rotation of humanoid robot TEO.*

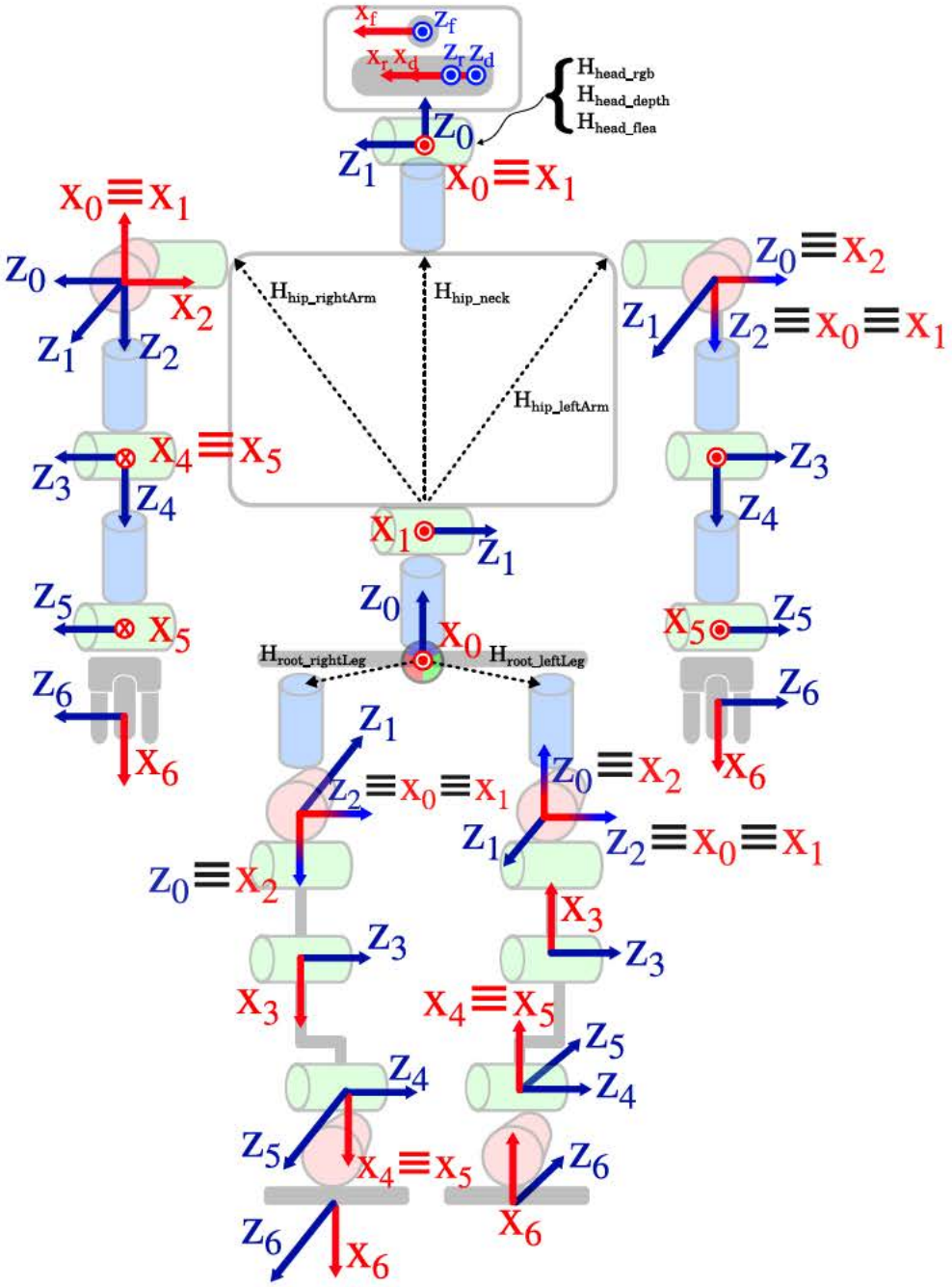

Figure B.2: *Denavit-Hartenberg directions of humanoid robot TEO.*

Limb	Link	θ (Z i-1)	D (Z i-1)	A (X i)	α (X i)
Lef leg	1	q7	0	0	90
	2	90+q8	0	0	90
	3	q9	0	-l10	0
	4	q10	-l16	-l11	0
	5	q11	0	0	90
	6	q12	0	-l12	0
Right leg	1	q6	0	0	90
	2	90+q5	0	0	90
	3	q4	0	l10	0
	4	q3	l16	l11	0
	5	q2	0	0	90
	6	q1	0	l12	0
Right arm	1	q15	0	0	90
	2	-90+q16	0	0	90
	3	-90+q17	l6	0	90
	4	q18	0	0	-90
	5	q19	l7	0	90
	6	90+q20	0	l8	0
Lef arm	1	q21	0	0	90
	2	90+q22	0	0	90
	3	90+q23	l6	0	90
	4	q24	0	0	-90
	5	q25	l7	0	90
	6	90+q26	0	l8	0
Trunk	1	q13	l0	0	-90
	2	q14	0	0	0
Head	1	q27	0	0	90
	2	q28	0	0	-90

from root

H_head_rgb	TrasZ(-l14) * TrasY(l3) * RotY(90) * RotZ(180)
H_head_depth	TrasZ(-l14-l15) * TrasY(l3) * RotY(90) * RotZ(180)
H_head_f ea	TrasY(l3+l4) * RotY(90) * RotZ(180)
H_hip_neck	RotX(90) * TrasZ(l1+l2)
H_hip_lef Arm	TrasY(-l1) * TrasZ(l5) * RotZ(90)
H_hip_rightArm	TrasY(-l1) * TrasZ(-l5) * RotZ(-90) * RotX(180)
H_root_lef Leg	TrasY(l13) * TrasZ(-l9) * RotZ(90)
H_root_rightLeg	TrasY(-l13) * TrasZ(-l9) * RotZ(90) * RotX(180)

Figure B.3: *Denavit-Hartenberg parameters of humanoid robot TEO.*

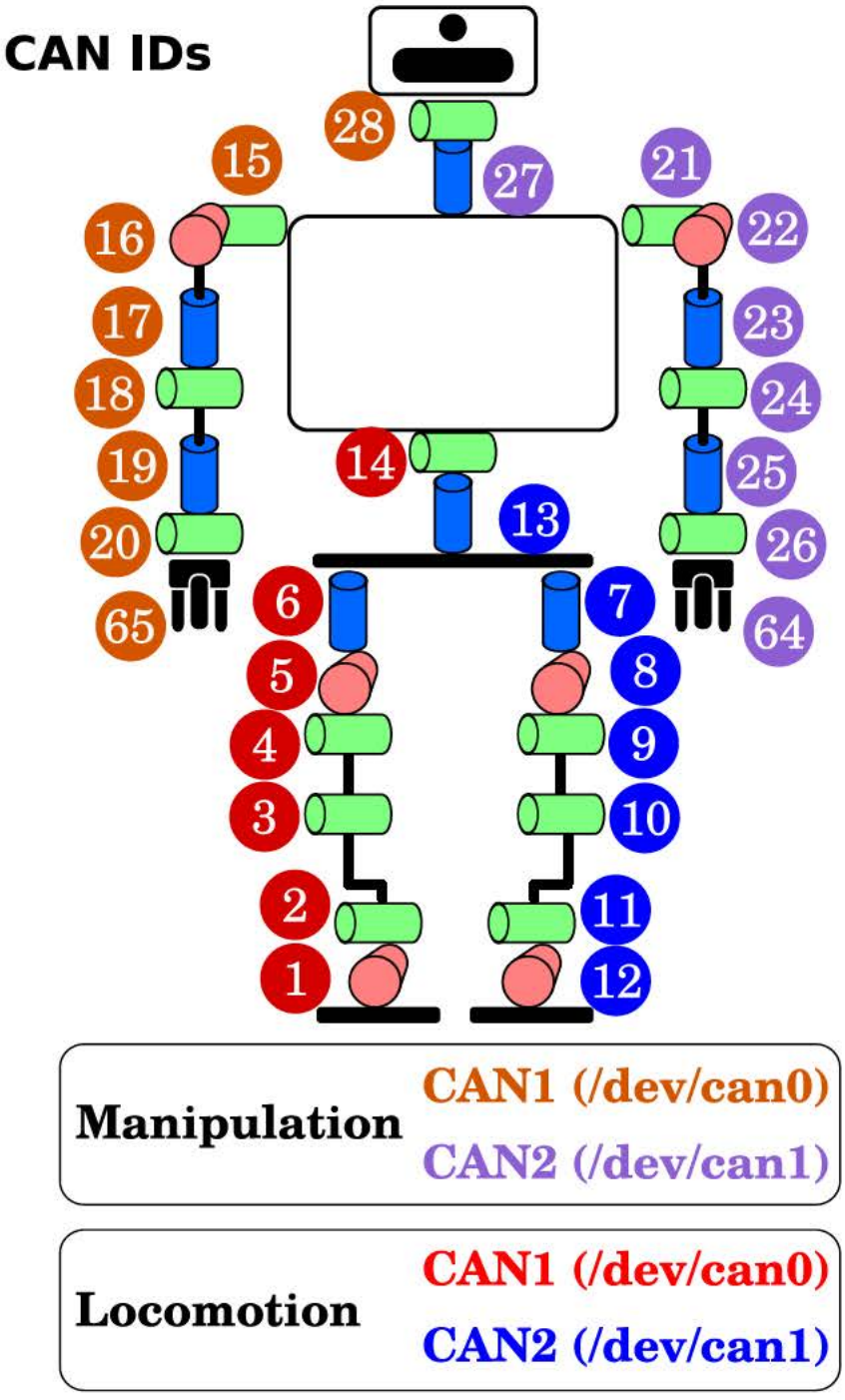

Figure B.4: *CAN (Controller Area Network) identification number of the joints of humanoid robot TEO.*

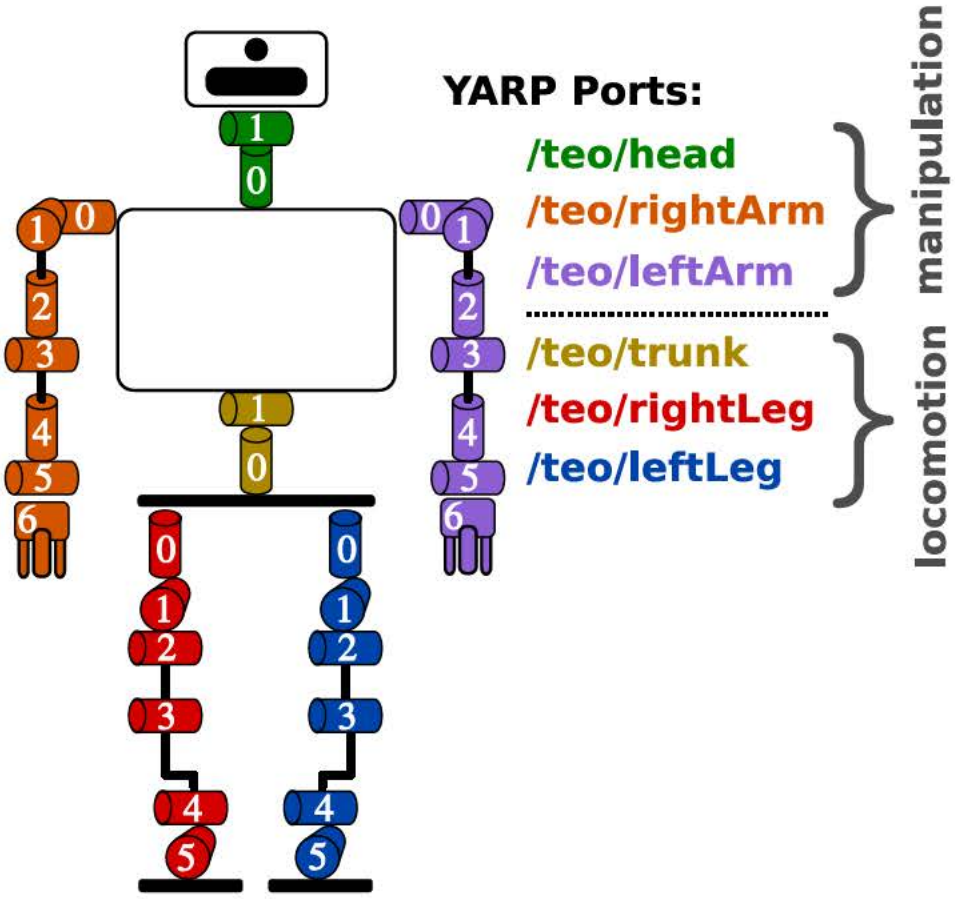

Figure B.5: *YARP identification number of the joints of humanoid robot TEO.*

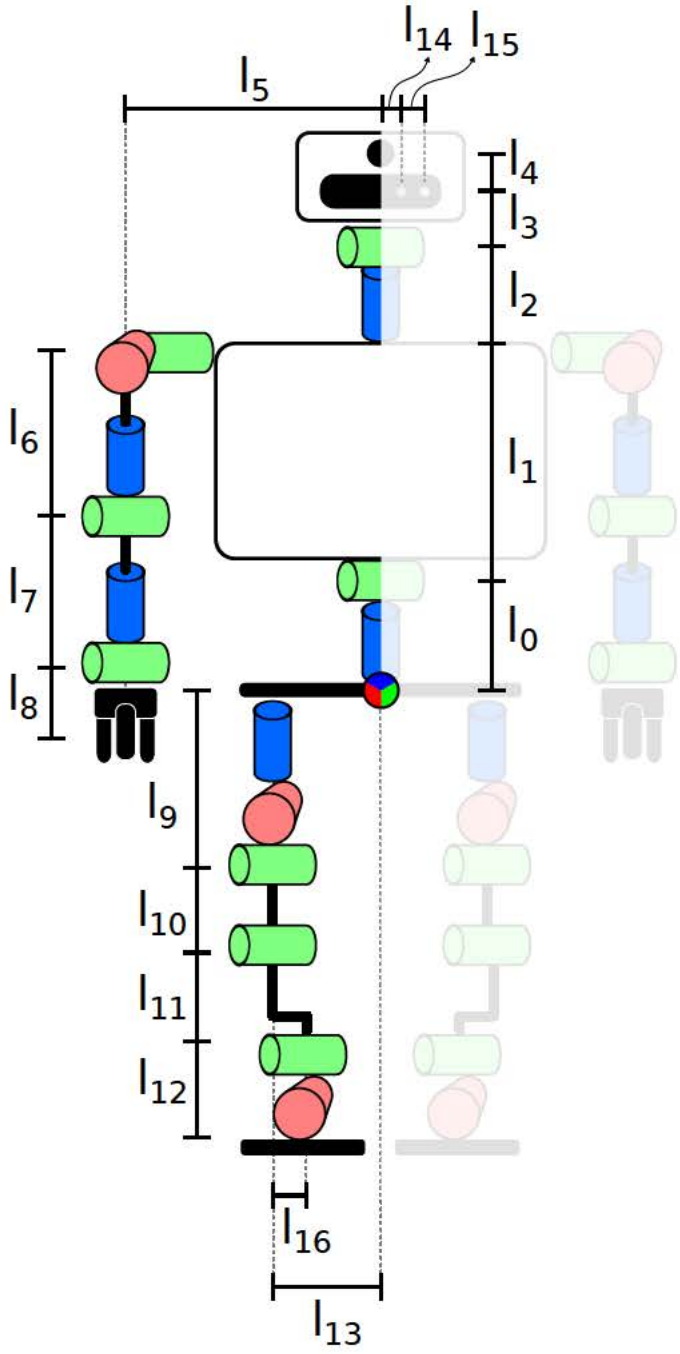

Figure B.6: *Mechanical links lengths distribution of humanoid robot TEO.*

lengths	distance [mm]
l0	191,7
l1	305
l2	161,3
l3	59,742
l4	37,508
l5	346,92
l6	329,01
l7	202
l8	187,496
l9	92
l10	330
l11	299,897
l12	123,005
l13	146
l14	18
l15	26
l16	17,52

Figure B.7: *Mechanical links lengths measurements of humanoid robot TEO.*

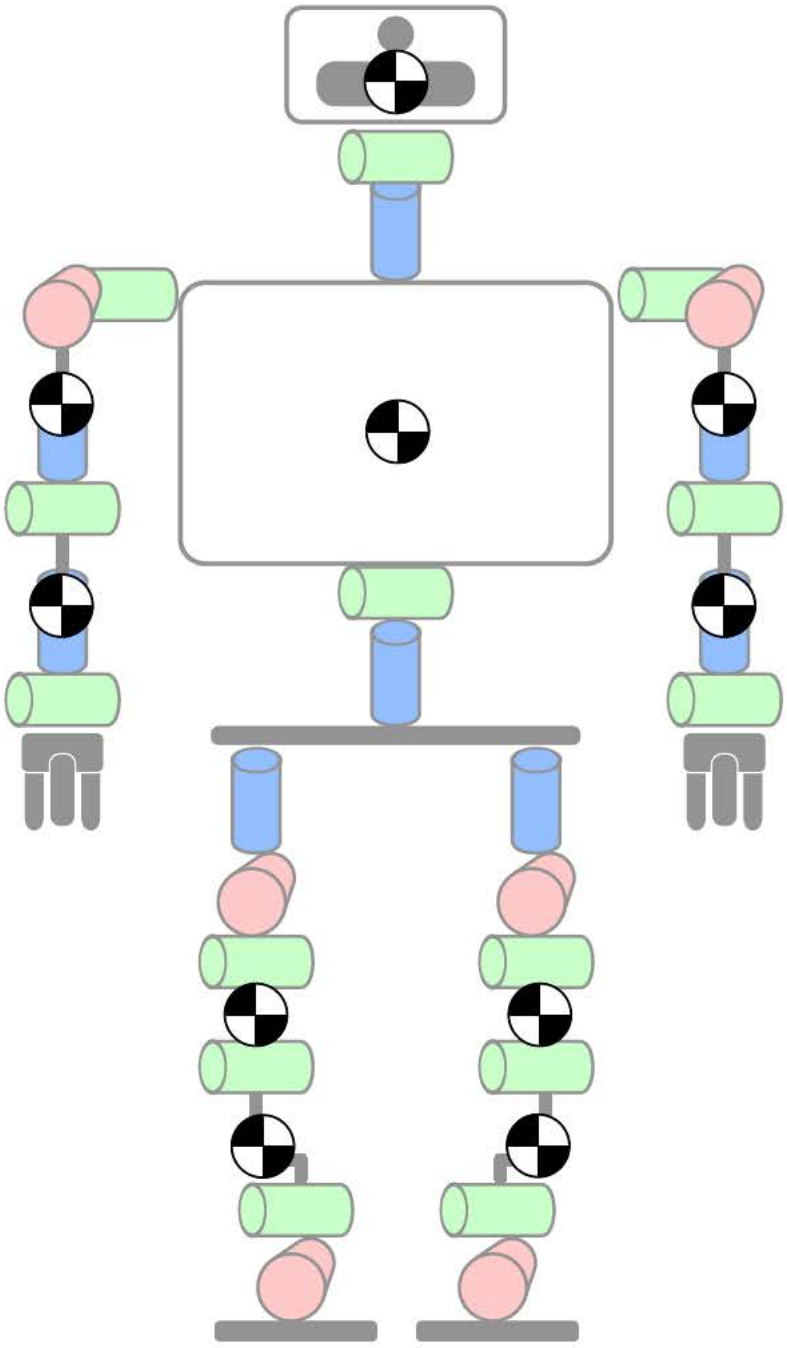

Figure B.8: *Position of the Center of Masses in the concentrated mass model of humanoid robot TEO.*

Bibliography

Akgun, B., Cakmak, M., Yoo, J. W., & Thomaz, A. L. (2012). 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* (pp. 391–398). (cited on p. 20.)

Albrecht, T. (2009). Dynamic time warping. In *Information retrieval for music and motion* (pp. 69–84). Springer. (cited on pp. 38 and 51)

Ali Paikan, G. M., Paul Fitzpatrick, & Natale, L. (2014). Data flow ports monitoring and arbitration. *Journal of Software Engineering for Robotics*, *5*(1), 80-88. (cited on p. 111.)

Alpaydin, E. (2014). *Introduction to machine learning*. MIT press. (cited on p. 37.)

An, C. H., Atkeson, C. G., & Hollerbach, J. M. (1988). *Model-based control of a robot manipulator* (Vol. 214). MIT press Cambridge, MA. (cited on p. 22.)

Argall, B. D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, *57*(5), 469–483. (cited on p. 19.)

Atkeson, C. G., & Schaal, S. (1997). Robot learning from demonstration. In *Icml* (Vol. 97, pp. 12–20). (cited on p. 16.)

Bekkering, H., Wohlschlager, A., & Gattis, M. (2000). Imitation of gestures in children is goal-directed. *The Quarterly Journal of Experimental Psychology: Section A*, *53*(1), 153–164. (cited on pp. 12 and 14)

Billard, A., Epars, Y., Calinon, S., Schaal, S., & Cheng, G. (2004). Discovering optimal imitation strategies. *Robotics and Autonomous Systems*, *47*(2), 69–77. (cited on pp. 9 and 14)

Billard, A. G., Calinon, S., & Guenter, F. (2006). Discriminative and adaptive imitation in uni-manual and bi-manual tasks. *Robotics and Autonomous Systems*, *54*(5), 370–384. (cited on p. 20.)

Bona, B., & Indri, M. (2005). Friction compensation in robotics: an overview. In *Ieee conference on decision and control, 2005 and 2005 european control conference. cdc-ecc'05. 44th* (pp. 4360–4367). (cited on p. 21.)

Bonaci, T., Herron, J., Yusuf, T., Yan, J., Kohno, T., & Chizeck, H. J. (2015). To make a robot secure: An experimental analysis of cyber security threats against teleoperated surgical robots.
(cited on p. 109.)

Calinon, S. (2009). *Robot programming by demonstration: A probabilistic approach*. EPFL Press. (cited on p. 33.)

Calinon, S., & Billard, A. (2004). Stochastic gesture production and recognition model for a humanoid robot. In *Proceedings of ieee/rsj international conference on intelligent robots and systems (iros 2004)* (Vol. 3, pp. 2769–2774). (cited on p. 10.)

Calinon, S., & Billard, A. (2005). Recognition and reproduction of gestures using a probabilistic framework combining pca, ica and hmm. In *Proceedings of the 22nd international conference on machine learning* (pp. 105–112). (cited on p. 10.)

Calinon, S., & Billard, A. (2007). Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the acm/ieee international conference on human-robot interaction* (pp. 255–262). (cited on p. 10.)

Calinon, S., D'halluin, F., Sauser, E. L., Caldwell, D. G., & Billard, A. G. (2010). Learning and reproduction of gestures by imitation. *IEEE Robotics & Automation Magazine*, *17*(2), 44–54. (cited on pp. 9, 20, and 48)

Calinon, S., Guenter, F., & Billard, A. (2005a). Goal-directed imitation in a humanoid robot. In *Proceedings of ieee international conference on robotics and automation, 2005 (icra 2005)* (pp. 299–304). (cited on p. 14.)

Calinon, S., Guenter, F., & Billard, A. (2005b). Goal-directed imitation in a humanoid robot. In *Proceedings of the 2005 ieee international conference on robotics and automation. icra 2005* (pp. 299–304). (cited on p. 17.)

Calinon, S., Guenter, F., & Billard, A. (2007). 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. (cited on pp. 10 and 81)

Canudas, C., Astrom, K., & Braun, K. (1987). Adaptive friction compensation in dc-motor drives. *IEEE Journal of Robotics and Automation*, *3*(6), 681–685. (cited on p. 21.)

Chernova, S., & Veloso, M. (2008). Multi-thresholded approach to demonstration selection for interactive robot learning. In *3rd acm/ieee international conference on human-robot interaction (hri)* (pp. 225–232). (cited on p. 17.)

Chivers, D. S. (2012). *Human action recognition by principal component analysis of motion curves* (Unpublished doctoral dissertation). Wright State University. (cited on p. 11.)

Coble, K., Wang, W., Chu, B., & Li, Z. (2010). Secure software attestation for military telesurgical robot systems. In *Military communications conference, 2010-milcom* (pp. 965–970). (cited on p. 112.)

Denning, T., Matuszek, C., Koscher, K., Smith, J. R., & Kohno, T. (2009). A spotlight on security and privacy risks with future household robots: attacks and lessons. In *Proceedings of the 11th international conference on ubiquitous computing* (pp. 105–114). (cited on p. 109.)

Dzung, D., Naedele, M., Von Hoff, T. P., & Crevatin, M. (2005). Security for industrial communication systems. *Proceedings of the IEEE*, *93*(6), 1152–1177. (cited on p. 112.)

Erlhagen, W., Mukovskiy, A., Bicho, E., Panin, G., Kiss, C., Knoll, A., ... Bekkering, H. (2006). Goal-directed imitation for robots: A bio-inspired approach to action understanding and skill learning. *Robotics and autonomous systems*, *54*(5), 353–360. (cited on p. 14.)

Friedman, J., Hastie, T., & Tibshirani, R. (2001). *The elements of statistical learning* (Vol. 1). Springer series in statistics Springer, Berlin. (cited on p. 38.)

Gallese, V., Fadiga, L., Fogassi, L., & Rizzolatti, G. (1996). Action recognition in the premotor cortex. *Brain*, *119*(2), 593–609. (cited on p. 12.)

Gomes, S. C. P., & Santos da Rosa, V. (2003). A new approach to compensate friction in robotic actuators. In *Proceedings ieee international conference on robotics and automation, 2003. icra'03.* (Vol. 1, pp. 622–627). (cited on p. 22.)

Groza, B., & Dragomir, T.-L. (2008). Using a cryptographic authentication protocol for the secure control of a robot over tcp/ip. In *Ieee international conference on automation, quality and testing, robotics. aqtr* (Vol. 1, pp. 184–189). (cited on p. 112.)

Guizzo, E., & Ackerman, E. (2012). The rise of the robot worker. *Spectrum, IEEE*, *49*(10), 34–41. (cited on p. 21.)

Gupta, V., Gupta, S., Chang, S., & Stebila, D. (2002). Performance analysis of elliptic curve cryptography for ssl. In *Proceedings of the 1st acm workshop on wireless security* (pp. 87–94). (cited on p. 113.)

Hans, M., Graf, B., & Schraft, R. (2002). Robotic home assistant care-o-bot: Past-present-future. In *Proceedings of 11th ieee international workshop on robot and human interactive communication* (pp. 380–385). (cited on p. 109.)

Ijspeert, A. J., Nakanishi, J., & Schaal, S. (2002). Movement imitation with nonlinear dynamical systems in humanoid robots. In *Ieee international conference on robotics and automation (icra)* (Vol. 2, pp. 1398–1403). (cited on p. 19.)

Javaid, A. Y., Sun, W., Devabhaktuni, V. K., & Alam, M. (2012). Cyber security threat analysis and modeling of an unmanned aerial vehicle system. In *Ieee conference on technologies for homeland security (hst)* (pp. 585–590). (cited on p. 108.)

Jetchev, N., & Toussaint, M. (2011). Task space retrieval using inverse feedback control. In *Proceedings of the 28th international conference on machine learning (icml-11)* (pp. 449–456). (cited on pp. 16 and 18)

Johnson, M., & Demiris, Y. (2004). Abstraction in recognition to solve the correspondence problem for robot imitation. In *Proceedings of towards autonomous robotic systems* (pp. 63–70). Springer. (cited on p. 14.)

Kermani, M. R., Wong, M., Patel, R. V., Moallem, M., & Ostojic, M. (2004). Friction compensation in low and high-reversal-velocity manipulators. In *Proceedings on ieee international conference on robotics and automation, 2004. icra'04.* (Vol. 5, pp. 4320–4325). (cited on p. 22.)

King, H. H., Tadano, K., Donlin, R., Friedman, D., Lum, M. J., Asch, V., ... Hannaford, B. (2009). Preliminary protocol for interoperable telesurgery. In *International conference on advanced robotics. icar 2009* (pp. 1–6). (cited on p. 109.)

Kober, J., Mohler, B., & Peters, J. (2008). Learning perceptual coupling for motor primitives. In *Ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 834–839). (cited on p. 19.)

Kober, J., Wilhelm, A., Oztop, E., & Peters, J. (2012). Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, *33*(4), 361–379. (cited on p. 19.)

Kostic, D., de Jager, B., Steinbuch, M., & Hensen, R. (2004). Modeling and identification for high-performance robot control: an rrr-robotic arm case study. *IEEE Transactions on Control Systems Technology*, *12*(6), 904–919. (cited on p. 22.)

Lee, G. S., & Thuraisingham, B. (2012). Cyberphysical systems security applied to telesurgical robotics. *Computer Standards & Interfaces*, *34*(1), 225–229. (cited on p. 112.)

Liu, M., & Quach, N. H. (2001). Estimation and compensation of gravity and friction forces for robot arms: Theory and experiments. *Journal of Intelligent and Robotic Systems*, *31*(4), 339–354. (cited on p. 22.)

Luca, A. D., & Panzieri, S. (1993). Learning gravity compensation in robots: Rigid arms, elastic joints, flexible links. *International journal of adaptive control and signal processing*, *7*(5), 417–433. (cited on p. 67.)

Luo, R. C., Yi, C., & Perng, Y. W. (2011). Gravity compensation and compliance based force control for auxiliarily easiness in manipulating robot arm. In *2011 8th asian control conference (ascc)* (pp. 1193–1198). (cited on p. 22.)

Mallon, N., van de Wouw, N., Putra, D., & Nijmeijer, H. (2006). Friction compensation in a controlled one-link robot using a reduced-order observer. *IEEE Transactions on Control Systems Technology*, *14*(2), 374–383. (cited on p. 94.)

Martin, P. K., & General, I. (2012). Nasa cybersecurity: An examination of the agency's information security. *US House of Representatives, Feb*. (cited on p. 109.)

Martínez, S., Monje, C. A., Jardón, A., Pierro, P., Balaguer, C., & Muñoz, D. (2012). TEO: Full-size humanoid robot design powered by a fuel cell system. *Cybernetics and Systems*, *43*(3), 163–180. (cited on pp. 74 and 86)

McClean, J., Stull, C., Farrar, C., & Mascareñas, D. (2013). A preliminary cyber-physical security assessment of the robot operating system (ros). In *Spie defense, security, and sensing* (pp. 874110–874110). (cited on p. 110.)

Metta, G., Fitzpatrick, P., & Natale, L. (2006). Yarp: yet another robot platform. *International Journal on Advanced Robotics Systems*, *3*(1), 43–48. (cited on pp. 80 and 110)

Mickle, P. (1961). *A peep into the automated future.* (cited on p. 1.)

Mohammad, Y., & Nishida, T. (2013). Tackling the correspondence problem. In *Active media technology* (pp. 84–95). Springer. (cited on p. 15.)

Mohanarajah, G., Hunziker, D., D'Andrea, R., & Waibel, M. (2015). Rapyuta: A cloud robotics platform. *IEEE Transactions on Automation Science and Engineering*, *12*(2),

481–493. (cited on pp. 108 and 112)

Morante, S. (2012). *Interfaz y librería para visión artificial, navegación y seguimiento en robótica* (Tech. Rep.). Leganés: Universidad Carlos III de Madrid. (cited on p. 80.)

Morante, S., Victores, J. G., & Balaguer, C. (2015a). Automatic Demonstration and Feature Selection for Robot Learning. In *Ieee international conference on humanoid robot - humanoids.* Seoul: IEEE. (cited on pp. 7 and 35)

Morante, S., Victores, J. G., & Balaguer, C. (2015b). Cryptobotics: Why Robots Need Cyber Safety. *Frontiers in Robotics and AI*, *2*(23), 1–6. doi: 10.3389/frobt.2015.00023 (cited on p. 7.)

Morante, S., Victores, J. G., Jardón, A., & Balaguer, C. (2014). Action effect generalization, recognition and execution through continuous goal-directed actions. In *Ieee international conference on robotics and automation (icra)* (pp. 1822–1827). (cited on pp. 6 and 34)

Morante, S., Victores, J. G., Jardón, A., & Balaguer, C. (2014). Action Effect Generalization, Recognition and Execution through Continuous Goal-Directed Actions. In *Ieee international conference on robotics and automation (icra).* (cited on p. 11.)

Morante, S., Victores, J. G., Jardón, A., & Balaguer, C. (2014). On using guided motor primitives to execute continuous goal-directed actions. In *Ro-man: The 23rd ieee international symposium on robot and human interactive communication* (pp. 613–618). (cited on pp. 7 and 37)

Morante, S., Victores, J. G., Jardón, A., & Balaguer, C. (2015). Humanoid robot imitation through continuous goal-directed actions: an evolutionary approach. *Advanced Robotics*, *29*(5), 303–314. (cited on p. 7.)

Morante, S., Victores, J. G., Martinez, S., & Balaguer, C. (2014). Sensorless friction and gravity compensation. In *14th ieee-ras international conference on humanoid robots (humanoids)* (pp. 265–265). (cited on p. 98.)

Morante, S., Victores, J. G., Martinez de la casa, S., & Balaguer, C. (2015). Force-Sensorless Friction and Gravity Compensation for Robots. In *Robot 2015: Second iberian robotics conference.* Lisbon: SPR/SEIDROB. (cited on p. 7.)

Muhlig, M., Gienger, M., Hellbach, S., Steil, J. J., & Goerick, C. (2009). Task-level imitation learning using variance-based movement optimization. In *Ieee international conference on robotics and automation. icra'09* (pp. 1177–1184). (cited on pp. 16 and 17)

Muhlig, M., Gienger, M., Steil, J. J., & Goerick, C. (2009). Automatic selection of task spaces for imitation learning. In *Ieee/rsj international conference on intelligent robots and systems. iros 2009.* (pp. 4996–5002). (cited on pp. 16 and 17)

Mülling, K., Kober, J., Kroemer, O., & Peters, J. (2013). Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, *32*(3), 263–279. (cited on p. 19.)

Na, J., Chen, Q., Ren, X., & Guo, Y. (2014, Jan). Adaptive prescribed performance motion control of servo mechanisms with friction compensation. *IEEE Transactions on Industrial Electronics*, *61*(1), 486-494. doi: 10.1109/TIE.2013.2240635 (cited on p. 22.)

Nehaniv, C. L., & Dautenhahn, K. (1999). Of hummingbirds and helicopters: An algebraic framework for interdisciplinary studies of imitation and its applications. In *Interdisciplinary approaches to robot learning* (Vol. 24, p. 136). World Scientific. (cited on pp. 10 and 12)

Newton, I., Bernoulli, D., MacLaurin, C., & Euler, L. (1833). *Philosophiae naturalis principia mathematica* (Vol. 1). excudit G. Brookman; impensis TT et J. Tegg, Londini. (cited on p. 34.)

Olsson, H., Åström, K. J., Canudas de Wit, C., Gäfvert, M., & Lischinsky, P. (1998). Friction models and friction compensation. *European journal of control*, *4*(3), 176–195. (cited on p. 21.)

Papadopoulos, E. G., & Chasparis, G. C. (2004). Analysis and model-based control of servomechanisms with friction. *Journal of dynamic systems, measurement, and control*, *126*(4), 911–915. (cited on p. 22.)

Peters, J., Kober, J., Mülling, K., Krämer, O., & Neumann, G. (2013). Towards robot skill learning: From simple skills to table tennis. In *Machine learning and knowledge discovery in databases* (pp. 627–631). Springer. (cited on p. 19.)

Peters, J., & Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks*, *21*(4), 682–697. (cited on p. 19.)

Pfleeger, C. P., & Pfleeger, S. L. (2002). *Security in computing*. Prentice Hall Professional Technical Reference. (cited on p. 107.)

Press, W. H. (2007). *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press. (cited on p. 49.)

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... Ng, A. Y. (2009). Ros:

an open-source robot operating system. In *Icra workshop on open source software* (Vol. 3, p. 5). (cited on p. 110.)

Rizzolatti, G., Fadiga, L., Gallese, V., & Fogassi, L. (1996). Premotor cortex and the recognition of motor actions. *Cognitive brain research*, *3*(2), 131–141. (cited on p. 12.)

Rochat, M. J., Caruana, F., Jezzini, A., Escola, L., Intskirveli, I., Grammont, F., ... Umiltà, M. A. (2010, August). Responses of mirror neurons in area F5 to hand and tool grasping observation. *Experimental brain research*, *204*(4), 605–16. doi: 10.1007/s00221-010-2329-9 (cited on p. 12.)

Saegusa, R., Metta, G., Sandini, G., & Natale, L. (2013). Developmental action perception for manipulative interaction. In *Ieee international conference on robotics and automation (icra)* (p. 4967-4972). (cited on p. 14.)

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, *3*(6), 233–242. (cited on pp. 10 and 18)

Schaal, S. (2006). Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines* (pp. 261–280). Springer. (cited on p. 33.)

Schaal, S., Ijspeert, A., & Billard, A. (2003). Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, *358*(1431), 537–547. (cited on p. 18.)

Sciavicco, L., & Villani, L. (2009). *Robotics: modelling, planning and control*. Springer. (cited on p. 67.)

Subramanian, K., & Suresh, S. (2012). Human action recognition using meta-cognitive neuro-fuzzy inference system. *International journal of neural systems*, *22*(06). (cited on p. 11.)

Tanenbaum, A. S. (2009). *Modern Operating Systems* (3rd ed.). Pearson-Prentice Hall. (cited on p. 108.)

Tarsitano, I. A.-A. (2012). Adjusting time series of possible unequal lengths. In *Sis. proceedings of the xlvi scientific meeting* (pp. 1–4). (cited on p. 16.)

Toris, R., Shue, C., & Chernova, S. (2014, April). Message authentication codes for secure remote non-native client connections to ros enabled robots. In *Ieee international conference on technologies for practical robot applications (tepra)* (p. 1-6). doi: 10.1109/TePRA.2014.6869141 (cited on p. 111.)

Traversaro, S., Del Prete, A., Muradore, R., Natale, L., & Nori, F. (2013). Inertial parameter identification including friction and motor dynamics. In *Ieee-ras international conference on humanoid robots (humanoid'13), atlanta, usa.* (cited on p. 22.)

Ugur, E., Sahin, E., & Oztop, E. (2012). Self-discovery of motor primitives and learning grasp affordances. In *Ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 3260–3267). (cited on p. 19.)

Victores, J. G., Morante, S., Jardón, A., & Balaguer, C. (2013a). Semantic action parameter inference through machine learning methods. In *Robocity2030 12th workshop robotica cognitiva.* (cited on p. 13.)

Victores, J. G., Morante, S., Jardón, A., & Balaguer, C. (2013b). Towards Robot Imagination Through Object Feature Inference. In *Proceedings of ieee/rsj international conference on intelligent robots and systems (iros 2013).* IEEE. (cited on p. 104.)

Virgala, I., & Kelemen, M. (2013). Experimental friction identification of a dc motor. *International Journal of Mechanics and Applications*, *3*(1), 26–30. (cited on pp. 68 and 95)

Vuong, T., Filippoupolitis, A., Loukas, G., & Gan, D. (2014). Physical indicators of cyber attacks against a rescue robot. In *Ieee international conference on pervasive computing and communications workshops (percom workshops)* (pp. 338–343). (cited on p. 110.)

Yamashita, Y., & Tani, J. (2008). Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment. *PLoS Comput Biol*, *4*(11), e1000220. (cited on p. 15.)