



Human-centric algebraic machine learning

Design of the benchmark tested for robotized ironing and garment folding - Final

Deliverable D7.6

WP7 - Use cases



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 952091.

Consortium



ALGEBRAIC AI



Chantalimaud
Foundation



EPROSIMA
The Middleware Experts



German
Research Center
for Artificial
Intelligence



TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

uc3m



Project

Title: ALMA: Human Centric Algebraic Machine Learning
Acronym: ALMA
Coordinator: Proyectos y Sistemas de Mantenimiento SL (eProsima)
Grant agreement: 952091
Call: H2020-EIC-FETPROACT-2019
Program: Horizon 2020
Start: 01 September 2020
Duration: 48 months
Website: <https://alma-ai.eu>
E-mail: alma@eprosima.com
Consortium: **Proyectos y Sistemas de Mantenimiento SL (eProsima) (EPROS)**, Spain
Champalimaud Foundation (CF), Portugal
ALGEBRAIC AI (ALG), Spain
German Research Center for Artificial Intelligence (DFKI), Germany
Inria (Inria), France
TU Kaiserslautern (TUK), Germany
FIWARE Foundation e.V. (FF), Germany
Technical Research Centre of Finland (VTT), Finland
Univ. Carlos III University (UC3M), Spain

Deliverable

Number: **D7.6**
Title: **Design of the benchmark tested for robotized ironing and garment folding - Final**
Month: 48
Work Package: WP7 - Use cases
Work Package leader: UNIVERSIDAD CARLOS III DE MADRID
Deliverable leader: UNIVERSIDAD CARLOS III DE MADRID
Deliverable type: DEM
Dissemination level: Dissemination Level
Date of submission: 2025-03-21
Version: v2.0
Status: Reviewing

Version history

Version	Date	Responsible	Author/Reviewer	Comments
v1.0	2025-02-23	CF/ALG	Fernando Martin-Maroto	Algebraic Methods Section Executive Summary marks to review. Review Sections and overall harmonization Major Review
v1.2	2025-02-24	UC3M	Alberto Jardón	
v2.0	2025-03-21	UC3M	Juan G. Victores, Francisco J. Naranjo-Campos, Bartek Lukawski, Ana Calzada	Minor Review
v2.1	2025-03-23	UC3M	Alberto Jardón	Added Summary of related code repositories, video of deliverable and dataset, reviewed section activities Changed figure Figure 23: Convolutional Neural Network.
v2.2	2025-04-10	UC3M	Juan G. Victores, Alberto Jardón	
v2.3	2025-04-29	UC3M	Juan G. Victores, Alberto Jardón	

Executive summary

This report corresponds to *Deliverable D7.6 - Design of the benchmark tested for robotized ironing and garment folding - Final* of the ALMA project. This deliverable focuses on developing a testbed prototype to analyze and compare the results of previous learning algorithms to human-centric AML-based approaches. Task 7.3 Higher-level cognition for domestic assistance robots involves the development of four validation environments in de-facto standard format, two of which serve as a baseline, and the rest focusing on a domestic assistance robot task.

Seven different environments have been developed within the scope of this project, each implemented as a Gymnasium (previously OpenAI Gym) environment. Gymnasium is a de-facto standard that includes simple pythonic interfaces capable of representing Markov Decision Processes (MDP), which is the general formalism used for representing Reinforcement Learning problems. It provides a thin layer of abstraction, enabling switching between environments while testing an algorithm or testing different algorithms against the same environment.

The first two environments are popular concepts in machine learning, but they have been reimplemented to “own” the code and enable seamless adaptation to any specific project requirement. There are other three environments are variations on the robotic ironing task, and the last two are dedicated to the problem of learning how to fold/unfold textiles. All of these environments adopt different levels of simplifications required to be able to apply the AML capabilities, as they have been progressively adapted to the specific capabilities needed to model RL problems. In this way, we finally adapt the specifications of the simulation environments to those needed for the work package objective of a domestic assistance robot task.

The benchmark tested for robotized ironing and garment folding includes **Bandit-v0**, **GridWorld-v0**, **FakeIroning-v0**, **TeoIroning-v0**, **TiagoIroning-v0**, **FakeFolding-v0** and **TiagoFolding-v0**. E.g. **Bandit-v0** implements an n-armed bandit problem, which is a classical reinforcement learning problem with a single state and a Gaussian distribution associated with each action. In each Ironing environment, the algorithm must determine the best policy, via learning through interaction, to achieve the highest reward in expectation. In each Folding environment, the algorithm must determine the best pick and place positions, to reduce error (in synthetic data situations where a Ground Truth exists) or Rectangularity (as a proxy to task achievement in real robot situations).

In conclusion, the D7.6 report the development of a benchmark for robotic ironing and garment folding, focusing on higher-level cognition for domestic assistance robots using a variety of environments and solver algorithms. The authors plan to improve the results and confidence levels of their algorithms by integrating them with AML-based approaches.

Contents

Executive Summary	4
Contents	5
Acronyms	6
1 Introduction	7
2 Robot Platforms	8
2.1 TEO Humanoid Robot	8
2.2 TIAGo++ Humanoid Robot	9
3 Robotized Ironing	13
3.1 Environments	14
3.1.1 Bandit	14
3.1.2 Grid World	15
3.1.3 Ironing: 2D (“Fake”)	15
3.1.4 Ironing: TEO	16
3.1.5 Ironing: TIAGo++	16
3.2 Observation Wrapper	17
3.2.1 Box to Discrete	17
3.3 Solver Algorithms	17
3.3.1 Baseline Algorithm: Cross-Entropy Method	18
3.3.2 AML Method: In-context learning	19
3.4 Experiments and Results	21
3.4.1 Bandit	21
3.4.2 Grid World	21
3.4.3 Fake Ironing, TEO and TIAGo++ Ironing	22
4 Robotized Folding	25
4.1 Dataset and Ground-Truth Visualizer	26
4.2 Environments	27
4.2.1 Folding: 2D (“Fake”)	28
4.2.2 Folding: TIAGo++	29
4.3 Solver Algorithms	31
4.3.1 Baseline Algorithm: Convolutional Neural Network	31
4.3.2 AML Methods: Regression	31
4.4 Experiments and Results	33
4.4.1 Fake Folding	33
4.4.2 TIAGo++ Folding	34
5 Summary of Activities	36
6 Summary of related code repositories and dataset	40
References	41

Acronyms

CEM	Cross-Entropy Method.
DoF	Degrees of Freedom.
IMU	Inertial Measurement Unit.
MDP	Markov Decision Processes.
MPC	Model Predictive Control.
MuJoCo	Multi-Joint dynamics with Contact.
ROS	Robot Operating System.
UC3M	Universidad Carlos III de Madrid.
YARP	Yet Another Robot Platform.

1 Introduction

The main developments of this deliverable of WP7 (Use cases) correspond to the development of a testbed prototype to analyse and compare the results of previous learning algorithms to AML-based [1] approaches. To implement that goal the following tasks have been completed:

- Task 7.3 Higher-level cognition for domestic assistance robots.

Highlights:

1. Development of seven validation environments in de-facto standard format.
2. Two of these environments serve to establish a baseline, while the rest focus on a domestic assistance robot task.
3. Development of an additional environment wrapper to provide a different observation space.
4. Experimental evaluation and validation of algorithms on environments.

For the experiments, the simulated and real robot have been used.

The humanoid robot TEO [2] from Universidad Carlos III de Madrid (UC3M) was one the selected robotic platform for the real robot experiments. TEO is a 60 kg humanoid robot with 28 Degrees of Freedom (DoF) and a height of 1.7 m. The robot trunk (torso) is composed by 2 DoF, and each of the arms of the robot is composed by 6 DoF with a three-finger robot hand. The humanoid robot simulator used for the experiments with TEO is OpenRAVE-based [3] with UC3M plugins¹. Communications with the simulator and the real robot are implemented via Yet Another Robot Platform (YARP) [4]. The other humanoid robot used for manipulation is the research-oriented TIAGo++ from PAL Robotics². Here, Multi-Joint dynamics with Contact (MuJoCo) simulations are used, as well as Robot Operating System (ROS) communications.

Crucially, to facilitate rapid iteration and avoid potential bottlenecks in our research pipeline during the critical sim-to-real transition phase, we maintained a backup robot platform. While our primary focus was on the TEO humanoid robot, we also had access to a TIAGO robot. This redundancy proved vital due to a fatal last-hour power electronics board failure in the TEO robot, which left it unavailable for the final months of the project.

Our algorithms have always demonstrated a high degree of platform independence. To capitalize on this inherent portability, we strategically selected both TEO and TIAGO, not only for their different morphologies and capabilities, but also because they both support ROS and YARP middleware. This commonality allowed us to seamlessly transfer our control policies between platforms, using the same framework to handle sensory input and command motor actuators. Having both TEO and TIAGO ensured that if TEO encountered unforeseen hardware or software problems that would significantly delay our experimental progress, we could quickly deploy and validate our algorithms on the TIAGO platform. This allowed for faster validation cycles and more robust algorithm development because we were able to test in different kinematic scenarios. For most of the experiments, the torso and one full arm of the robot were used, while keeping the rest of the robot static.

¹<https://github.com/roboticslab-uc3m/openrave-yarp-plugins>

²See <https://blog.pal-robotics.com/tiago-bi-manual-robot-research/>, accessed on March 19, 2025.

2 Robot Platforms

The following Humanoid Robot platforms have been used: TEO and TIAGo++.

2.1 TEO Humanoid Robot

The humanoid robot TEO [2] from UC3M was the selected robotic platform for the Robotized Ironing real experiments. TEO, as seen in Figure 1a, is a 60 kg humanoid robot with 28 DoF and a height of 1.7 m. The robot trunk (torso) is composed by 2 DoF, and each of the arms of the robot is composed by 6 DoF with a three-finger underactuated 1 DoF gripper. TEO's head is capable of pan and tilt motions, i.e. the neck has 2 DoF, and it hosts a camera and a depth sensor. On the wrists and ankles, TEO has 6-axis force-torque sensors, and the torso features an Inertial Measurement Unit (IMU). These peripherals enable the integration of vision- and force-based applications, for instance, object recognition, compliant manipulation and gait.

Figure 1b describes the coordinate frames according to the Denavit-Hartenberg convention that allow to obtain the description of the kinematic tree. Task space-related applications rely on these calculations to transform joint angles into positions and orientations of the desired robot limbs (feet and hands).

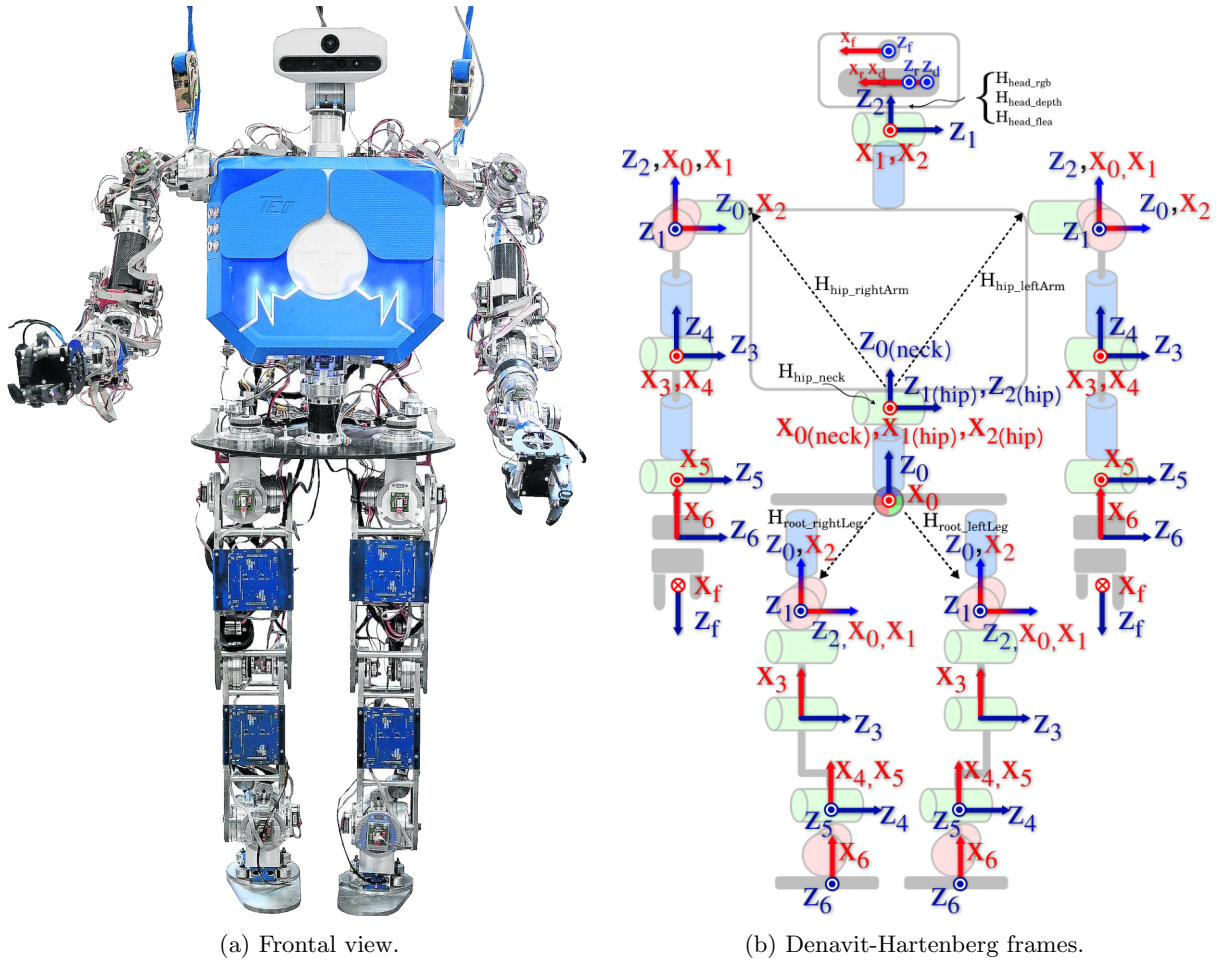


Figure 1: TEO humanoid robot.

Figure 2 depicts a sample controller architecture for YARP-based applications on TEO. This middleware and framework facilitates the adoption of distributed computing across multiple nodes that coexist in a local network. These nodes can be grouped in layers such as direct actuation, joint control, cartesian control, and external sensors and input devices. Commands can be centralized in a single controller unit as depicted, or split over multiple nodes to monitor internal and external conditions such as stability during walking, obstacle avoidance, etc. Compatibility with the ROS 2 middleware is also possible to interact between these two networks, and to integrate third-party components in our own architectures.

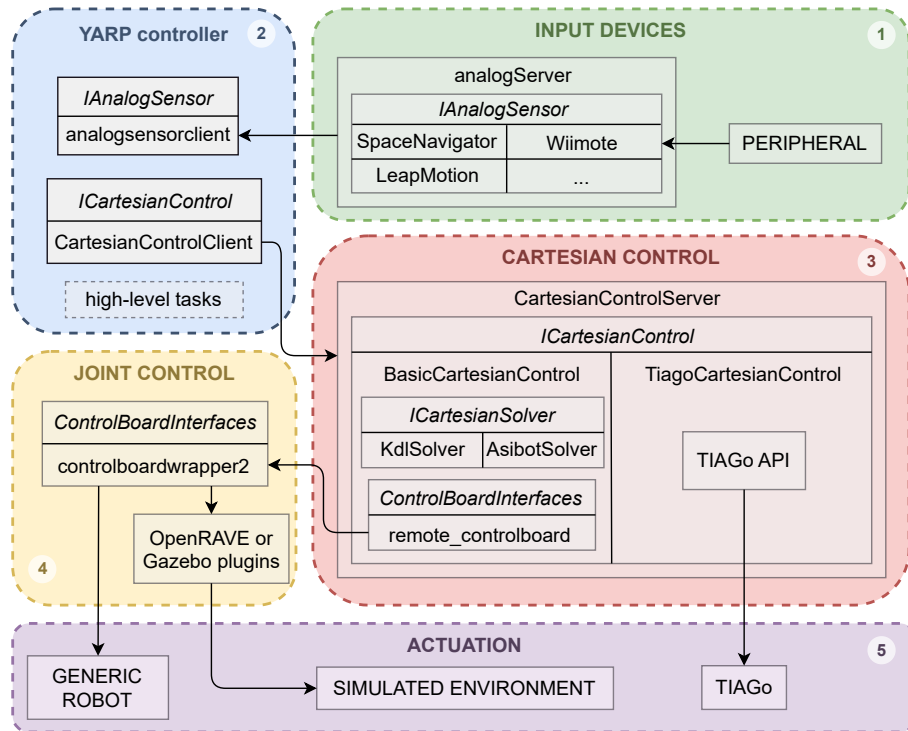


Figure 2: YARP-based controller architecture.

2.2 TIAGo++ Humanoid Robot

The other humanoid robot used for manipulation is the research-oriented TIAGo++ from PAL Robotics³, as seen in Figure 3. It has a modular design and can be adapted to research needs by customizing it with add-ons. In its basic version, it is composed of a mobile base, a torso, two arms with their wrists and end-effectors and a head.

TIAGo++ features a mobile base with differential wheel with laser sensor ensuring smooth, stable and autonomous navigation. The PBM-2 base can be used as an independent mobile robot. Its laser sensors, situated on the sides, enable it to perform SLAM (Simultaneous Localization And Mapping). Additionally, it includes three ultrasonic sensors on the back to prevent crashing as it moves backwards. It can reach a maximum speed of 1.5 m/s and move around overcoming bumps of up to 1.5 cm. The maximum payload it can carry on top of itself is 100 kg.

³See <https://blog.pal-robotics.com/tiago-bi-manual-robot-research/>, accessed on March 19, 2025.



Figure 3: TIAGo++ two arms humanoid Robot .

For manipulation tasks, TIAGo++ is equipped with two robotic arms with seven degrees of freedom each, a prismatic torso, and grippers as end-effectors. Since TIAGo is designed to be highly modular, the arms can be de-attached from the torso, leaving the robot with one arm or with none, depending on the research necessities. The 7 DOF in TIAGo++ arms are distributed along the arm, with the first 4 DOF belonging to the arm and hosting brushless motors and the remaining three belonging to the wrist. Additionally, a force-torque sensor is fixed to the end-effector link, which is useful for detecting contacts during manipulation and thus making the arm compliant. Without the end effector, the arm can lift up to 3 kg. Two different end effectors, provided by PAL, can be mounted at the end of the wrist: the Hey5 hand and a gripper with parallel plates. Unofficial end effectors should be connected to the user panel using external cables. In this case, the parallel grippers are used. They are designed with a rubberized surface to enhance grip and minimize slippage when handling objects. This feature is particularly useful for tasks such as picking, placing, and manipulating items, ensuring a firm and secure hold.

Moreover, the robot TIAGo features an Orbbec Astra RGB-D camera, which can be seen in Figure 4. The sensor is mounted on a pan-and-tilt system located at the head position, providing comprehensive and effective perception of the environment. The images can be visualized using RVIZ⁴ or ROS image subscribers. Additionally, the head is built so that more sensors can be attached to its flat top. This combination of advanced grippers and perception capabilities makes the TIAGo++ robot well-suited for a wide range of complex manipulation tasks.

The robot uses Ubuntu Linux LTS as its operative system, and is entirely based on the ROS Melodic⁵ framework, providing a flexible and widely-used platform for robotic development. TIAGo++ is compliant with the `ros_control` package⁶ (Figure 5) which allows it to use predefined controllers or write custom ones. By default, the robot's base can receive velocity commands via a specific topic, as well as each

⁴See <http://wiki.ros.org/rviz>, accessed on March 20, 2025.

⁵See <https://www.ros.org/>, accessed on March 19, 2024.

⁶See http://wiki.ros.org/ros_control, accessed on March 20, 2025.



Figure 4: TIAGo++ head with RGB-D camera.

joint in the torso, head, arm, hand or gripper. Additionally, it can receive complete sets of waypoints and execute complete joint-space point-to-point trajectories. Finally, it can also play predefined motions described as a set of points for each joint to reach in specific time intervals, or load custom-written controllers.

On the other hand, the robot's complete kinematic chain is available in URDF format⁷, facilitating forward and inverse kinematics computations to achieve desired end-effector poses. This is accomplished using the PyKDL library⁸, a Python wrapper for the KDL (Kinematics and Dynamics Library) from the Orocos project. PyKDL provides efficient tools for kinematic calculations, including Jacobian computations and dynamic modeling, and is widely integrated with ROS for precise control of complex kinematic chains.

⁷See <http://wiki.ros.org/urdf>, accessed on March 19, 2025.

⁸See <https://docs.ros.org/en/diamondback/api/kdl/html/python/>, accessed on March 19, 2025.

ROS Control

Data flow of controllers

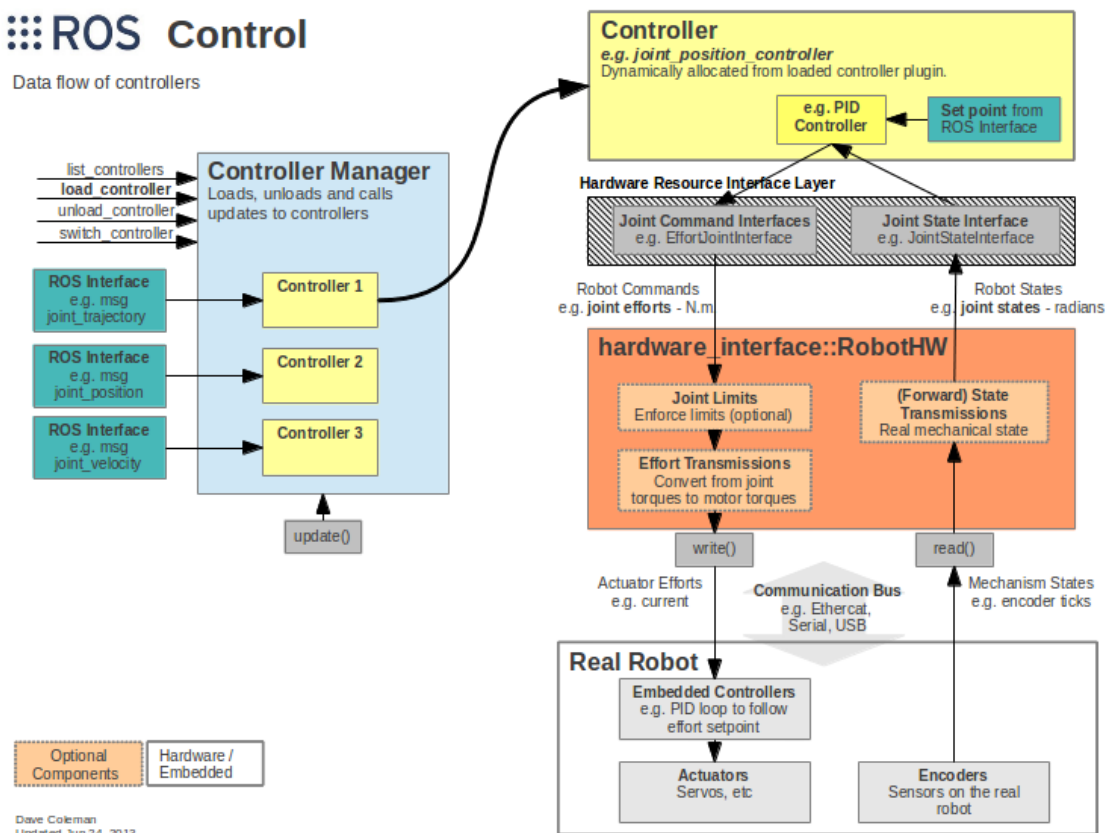


Figure 5: ROS control framework.

3 Robotized Ironing

Ironing is a complex task for a robot, which requires both a precise perception of the garment surface to detect wrinkles and a careful manipulation of the iron to apply enough force to remove them.

In this sense, it can be divided into perception and manipulation.

- Perception: Algorithms and precision to detect wrinkles of the garment surface. In this project, we leverage from previous developments [5] which provide the agent with an observation that has been transformed from a depth point cloud in color (via registration/alignment) in camera coordinates to a very simplified flat matrix, a.k.a the (rectified depth-map) image space for Robotized Ironing.

Figure 6 depicts the pipeline from full depth point cloud in color to the segmented garment, the equivalent to background removal in 3D.

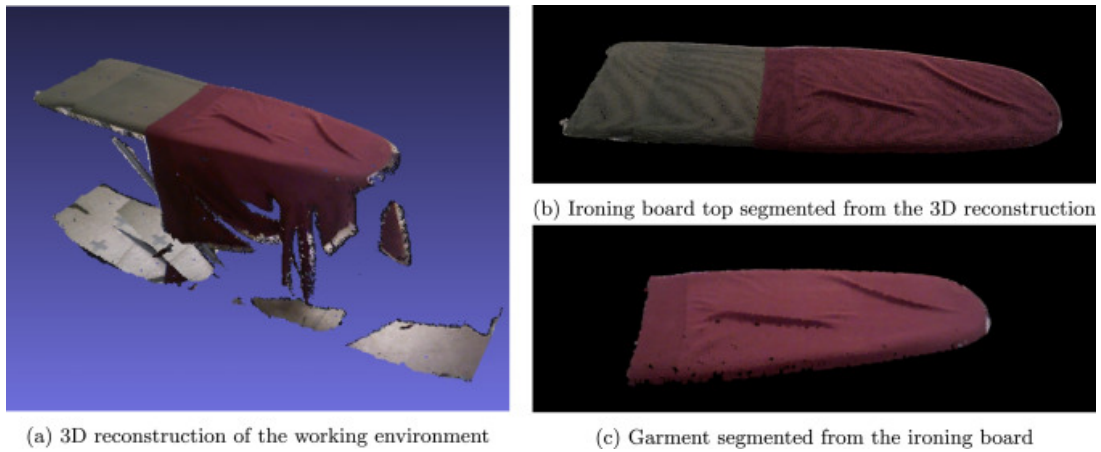


Figure 6: Pipeline from full color depth point cloud to the segmented garment (background removal) [5].

With respect to the previous figure, it is notable that 3D reconstruction (in the sense of merging several frames (e.g. via Kinect Fusion) is less important nowadays. Figure 7 depicts the full depth point cloud in color obtained from a single RGB-D frame, which is easily rotated, segmented (background removal) and binarized.

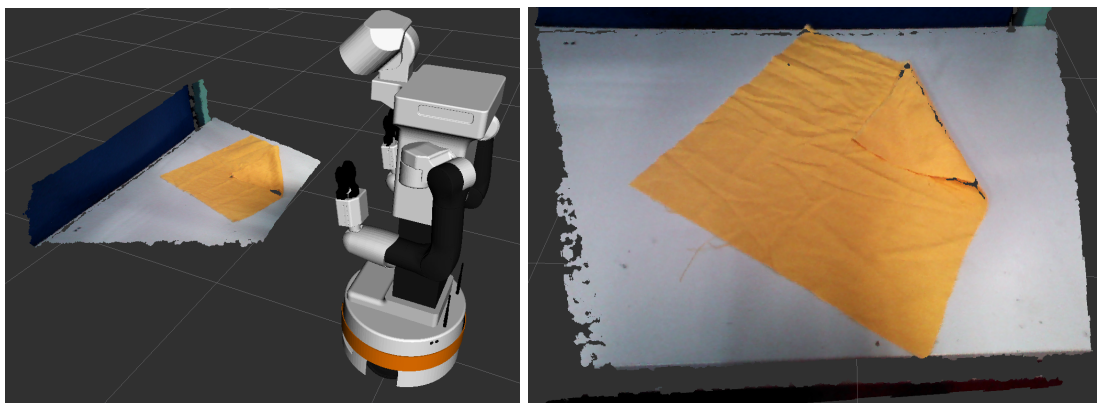


Figure 7: Depth point cloud in color as perceived from a single RGB-D frame of the TIAGo++ humanoid robot sensor.

All transformations are computed as part of the environments, namely (and with parameters specific to each camera and robot):

- From the pixels in image space to the coordinates in the SE(3) space of the camera frame (via camera intrinsic parameters).
- From the coordinates in the SE(3) space of the camera frame to the coordinates in the SE(3) space of the robot root frame (via camera extrinsic parameters as well as robot joint positions and kinematic model).

The binarization for this Robotized Ironing is such that garment surface is provided to the agent as a flat binary relatively low-resolution image, where each large pixel corresponds to a binary value that marks if the region contains a wrinkle or not (thresholding on the WiLD descriptor of [5]), which are images similar to those of Figure 9 thus rendering all developed environments of equal observation space.

- **Manipulation:** Computation of trajectories based on perception, in addition to handle the iron with care to apply enough force to remove the wrinkles. The robot end-effector arrives to the surface, where action space is limited to 8 actions on the surface (UP, UP_RIGHT, RIGHT, DOWN_RIGHT, DOWN, DOWN_LEFT, LEFT, UP_LEFT), following a similar scheme to 8-neighborhood pixel connectivity. Once the task has been accomplished (episode: done), the end-effector is retrieved. All transformations are computed as part of the environments, namely from the targets in the SE(3) space of the robot root frame to the actuators which are internally commanded in joint space (via the specific robot kinematics and forward/inverse kinematics and path generation for the robot end-effector).

3.1 Environments

Five different environments have been developed within the scope of this task, each implemented as a Gymnasium⁹ (previously OpenAI Gym) environment. Gymnasium is currently a de-facto standard which includes simple pythonic interfaces that are capable of representing MDP, which is the general formalism used for representing Reinforcement Learning problems. It provides a thin layer of abstraction, such that the environments are cleanly decoupled from the algorithms in the code. This enables switching between environments while testing an algorithm, as well as testing different algorithms against the same environment.

These environments are generic and reusable in terms of solver algorithms. To enhance reusability of code and algorithms, they have all been designed to share common observation spaces as well as action spaces. Whereas the first two environments are popular concepts in the area of machine learning (specifically, among reinforcement learning practitioners), they have been reimplemented to “own” the code and enable seamless adaptation to any specific project requirement. The other two environments are variations on the robotic ironing task, which matches the work package objective of a domestic assistance robot task.

Finally, they have been released and open-sourced under a common repository: <https://github.com/roboticslab-uc3m/gymnasium-alma>

3.1.1 Bandit

The environment **Bandit-v0** implements an n -armed bandit problem (also known as k -armed or simply multi-armed), which is a classical reinforcement learning problem. As in the original problem statement,

⁹<https://github.com/Farama-Foundation/Gymnasium>, accessed on March 19, 2025.

there is a single state (thus, can also be framed as a stateless environment), and the `n_arms` parameter can be used to establish the number of available actions. The reward associated to each action follows a Gaussian distribution centered at a value that is randomly assigned upon initialization (environment `reset()`). The observation space and the action space are **Discrete** (directly, without wrappers). This environment is headless (no visualization).

The algorithm must determine the best policy, which in this single state environment is given by the action of highest mean reward.

3.1.2 Grid World

Grid world environments are also conventional developments within the area of machine learning and a well known scenario among reinforcement learning practitioners. The environment `GridWorld-v0` implements one of such, a 2D grid world, depicted in Figure 8 (with activated PyGame graphics, also provides alternative headless and pure text modes). The algorithm must determine the best policy for the agent (small red box) to avoid wall obstacles (white) and reach the target (green larger box).



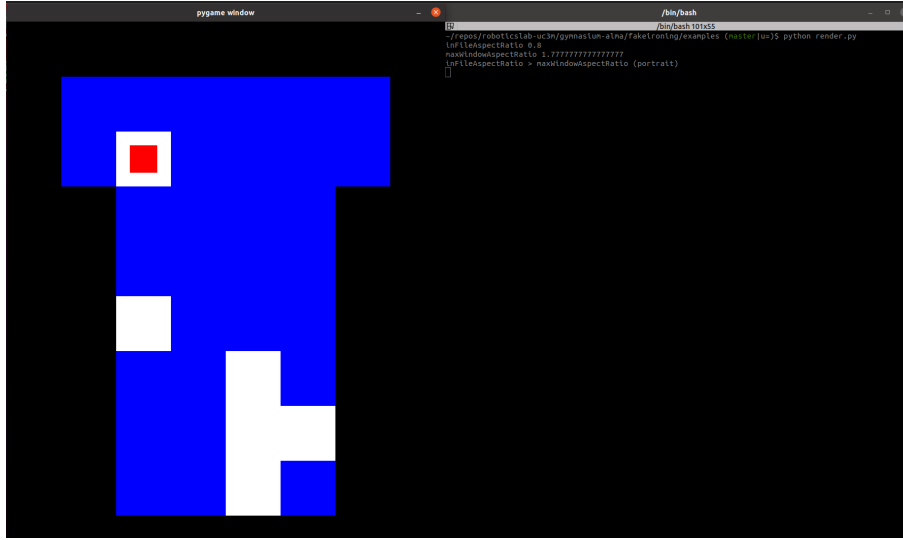
Figure 8: Screenshot of `GridWorld-v0`.

While several alternatives exist, this specific implementation is programmed to load the map configuration from a comma separated value (.csv) file. The other environments leverage from this concept, and build upon the same **Box** observation space and **Discrete** action space.

3.1.3 Ironing: 2D (“Fake”)

The `FakeIroning-v0` environment provides a simplified version of the ironing robotic task, which has been discretized and adapted to resemble the grid world, see Figure 9 (with activated PyGame graphics, also additionally provides alternative headless and pure text modes).

Here, three types of objects compose the environment: non-textile (black), textile without wrinkles (white), and textile with wrinkles (blue). The algorithm must determine the best policy for the agent (small red box) to avoid non-textile states and reach the target wrinkles. Textile states without wrinkles are considered neutral in terms of reward.

Figure 9: Screenshot of **FakeIroning-v0**.

The map configuration is also set up to be loaded from a comma separated value (.csv) file. The observation space is again **Box** and the action space remains **Discrete**.

3.1.4 Ironing: TEO

The **TeoIroning-v0** environment provided the initial instance of the ironing task, and can connect to the simulation environment (see Figure 10) as well as to the real TEO full-sized humanoid robot (see Figure 11). This is achieved by using the underlying communication middleware (YARP). Internally, it connects both to the joint-space level command robot controllers, as well as to an intermediate higher level controller which implements robot kinematics. The used kinematic chain includes both the trunk (2 DoF) and the left arm (6 DoF). This results in a redundant chain to achieve Cartesian-space position and orientation, which is treated by the intermediate controller at the kinematics solver level.

Most of the premises are similar to that of the **FakeIroning-v0** environment. The algorithm must determine the best policy for the embodied agent (robot trunk and limb movements that affect end-effector) to avoid non-textile states and reach the target wrinkles. Textile states without wrinkles are considered neutral in terms of reward. The map configuration is also set up to be loaded from a comma separated value (.csv) file. The observation space is again **Box** and the action space remains **Discrete**.

3.1.5 Ironing: TIAGo++

The **TiagoIroning-v0** environment implements the ironing task, building upon the architecture of **FakeIroning-v0**, similar to **TeoIroning-v0**. It is capable of connecting to the real TIAGo++ humanoid robot, as shown in Figure 12.

This functionality is achieved through the robot's kinematic chain and the PyKDL tool. The actions are converted into incremental adjustments to the end-effector's pose, which are then mapped to joint values via inverse kinematics. These joint values are subsequently transmitted to the real robot using ROS communication.

The rest of the environment follows the same structure as **TeoIroning-v0**. The algorithm must determine the optimal policy for the embodied agent (i.e., the robot's right-arm movements that influence the end-

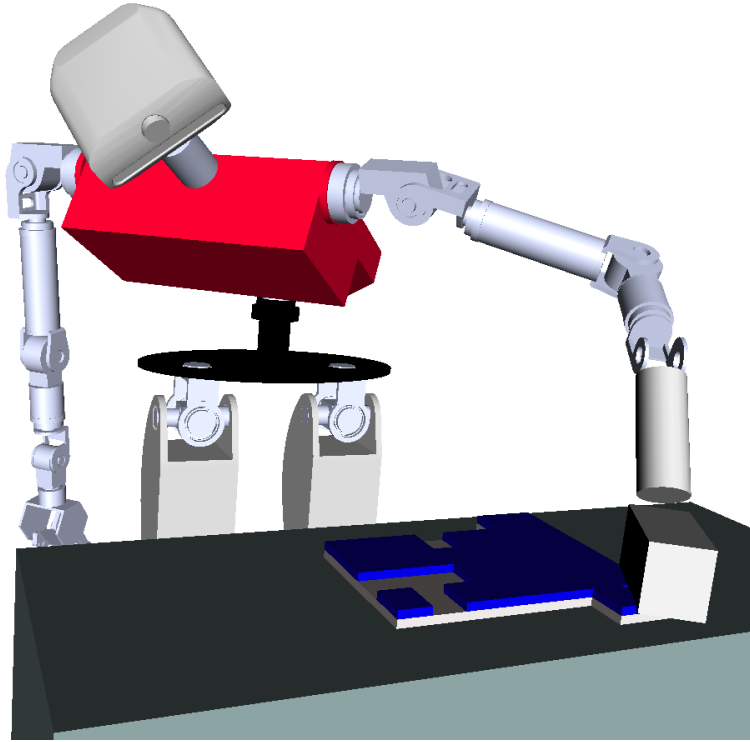


Figure 10: Screenshot of TeoIroning-v0 connected to TEO humanoid robot simulator.

effector) to avoid non-textile states and reach the target wrinkles. The map configuration is also loaded from a comma-separated values (.csv) file. The observation space remains a **Box**, and the action space continues to be **Discrete**.

3.2 Observation Wrapper

A small Gymnasium environment wrapper has been developed, which implements a transformation on the observation space.

3.2.1 Box to Discrete

The **BoxToDiscreteObservation** environment wrapper transforms any **Box** (e.g. 2D) observation space into a **Discrete** observation space (1D). It is matricially equivalent to a **flatten()** or **ravel()** operation, provided in other machine learning frameworks, but additionally implements the required class conversion as a side-effect. This wrapper is useful for testing environments such as **GridWorld-v0**, **FakeIroning-v0**, **Ironing-v0**, which at origin are implemented with 2D **Box** observation spaces, with reinforcement learning tabular or other methods which may require 1D **Discrete** observation spaces.

3.3 Solver Algorithms

For Robot Ironing, two solver algorithms were implemented. The implementation code of this section is provided in the following repository: <https://github.com/roboticslab-uc3m/alma-aml> that contains the Algebraic Machine Learning (AML) applied algorithms.

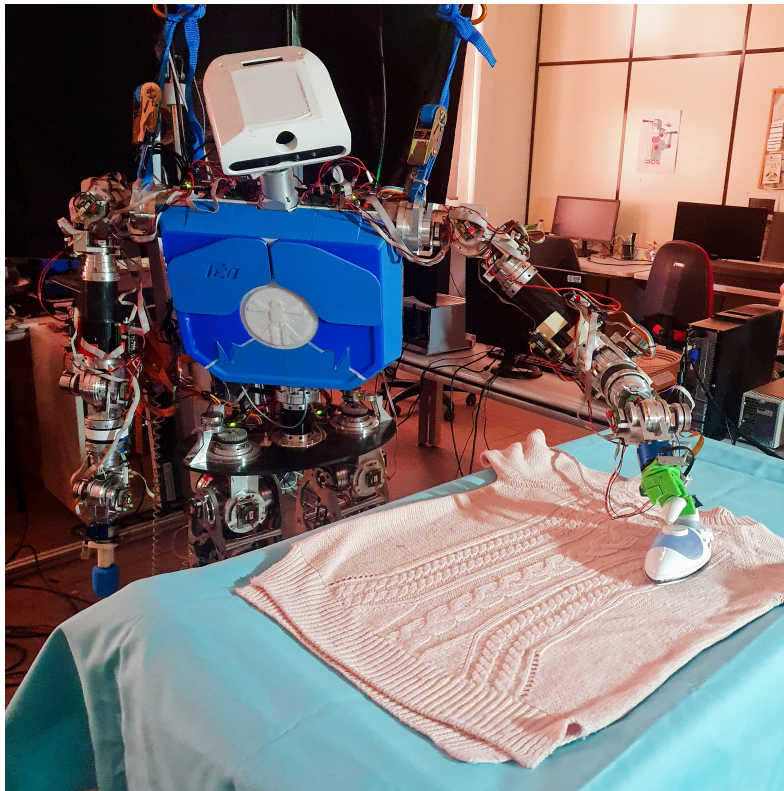


Figure 11: TeoIroning-v0 connected to TEO humanoid real robot.

3.3.1 Baseline Algorithm: Cross-Entropy Method

While methods like Deep Q-Networks (DQN) and Advantage Actor-Critic (A2C) are well-known and widely used in the reinforcement learning (RL) community, Cross-Entropy Method (CEM) is less commonly discussed, though it offers unique advantages [6]. Firstly, the cross-entropy method is really simple, which makes it an easy method to follow. For example, its implementation on PyTorch is less than 100 lines of code. Secondly, the method has good convergence. In simple environments that do not require complex, multistep policies to be learned and discovered, and that have short episodes with frequent rewards, the cross-entropy method usually works very well. Of course, lots of practical problems do not fall into this category, but sometimes they do. In such cases, the cross-entropy method (on its own or as part of a larger system) can be the perfect fit.

Building on this foundation, the “Elite Sample” method was introduced. A tabular approach using state-action-reward triplets could have addressed the problem at hand. However, our focus was to remain closer to reinforcement learning principles, emphasizing interaction-based learning by prioritizing the state-action pairs weighted by their associated rewards.

With this concept in mind, the notion of “elite samples” was borrowed from Cross-Entropy Method (CEM) literature [7]. A sample is a state-action pair, which proceeds from online interaction, and is originally stored with its associated obtained reward. “Elite samples” are obtained as the top-tier (set to 70 % in this experiment) per batch of these samples when ranked by obtained reward. In this setup, the AML engine is set in supervised mode as in subsection 3.4.1, but only provided with the “elite samples” for training per batch.



Figure 12: TeoIroning-v0 connected to TEO humanoid real robot.

3.3.2 AML Method: In-context learning

At the beginning of the ALMA project, Algebraic Machine Learning (AML) [1], had demonstrated effectiveness primarily in supervised learning tasks, such as handwritten character recognition. To apply AML to robotics, we thought that it was necessary to expand the existing AML methods to address unsupervised learning. Additionally, a method capable of addressing the challenges typically resolved by reinforcement learning was also desirable. As a result, we developed In-context Learning, an AML embedding method, which we tested in game-playing scenarios, such as tic-tac-toe and in a simulated robot navigation task.

While in the context of applying AML to reinforcement learning tasks, this is closer to what we could consider “Path planning” method in robotics. While a Grid World setup is a conventional reinforcement learning problem, one can also frame the problem from the motion planning perspective, resulting in a search problem. While this is mathematically an NP-hard problem, the reduced size of the search space as established enables feasibility of execution times. This practice is in line with practices such as Model Predictive Control (MPC) [8], where a motion planning at each control step can be implemented as long as it is sufficiently rapid. In this setup, the AML engine is set in this mode where the embeddings attempt to solve the path planning problem.

Given a board state in a game, it is usually possible to have more than one valid move. In statistical learning methods, this scenario does not pose a problem; however, in Algebraic Machine Learning, it may lead to inconsistencies depending on the embedding used. For instance, if the term B represents the state of the board, the axioms $move_1 \leq B$ and $move_2 \leq B$ may result in an inconsistent axiomatization, depending upon other axioms employed that use negation.

In-context Learning aims to resolve this issue by relying on the concept of *context constants* introduced in [9]. The principal approach involves employing axioms that are valid within a given context, for example, a particular game or even a specific move. The axioms would thus be formulated as:

$$\begin{aligned}
 context_A \odot move_1 &\leq B \odot context_A \\
 context_B \odot move_2 &\leq B \odot context_B \\
 context_C \odot move_3 &\leq B \odot context_C \\
 &\vdots
 \end{aligned}$$

The primary challenge was of a computational nature, as the number of context constants could grow to a very large number. To address this issue, the engine had to be modified. This modification involved finding alternatives to certain loops that initially iterated over the set of constants. Once the problem was overcome, we could test the approach with tic-tac-toe in 3×3 and 4×4 boards.

The algebra was initially trained in a supervised manner by observing tic-tac-toe games. Since an optimal tic-tac-toe policy never loses, the policy employed was near-optimal, allowing the algebra to learn how to win the game when the opponent makes a mistake. The players were programmed to make the second-best move with a probability of 10 percent.

An algebraic model M was derived from this training, using the context constants. To determine the optimal move given the state B of the board, we selected for the algebra the move with the smallest discriminant (see [10]). The move policy for the algebra was given by:

$$x = \operatorname{argmin}_i |dis_M(move_i, B)|,$$

where the algebra responds with $move_x$.

This method resulted in an algebraic model capable of winning against a bot implementing a 90 percent optimal policy. This promising result suggests that AML can effectively play games and also that uncertainty may be addressable using AML methods.

To apply this methodology to robotics, we aimed to implement In-context Learning in an unsupervised manner. For this purpose, we examined the problem of robot navigation in a grid world.

The input for this navigation problem consisted of a grid of either black or white cells. A black cell cannot be traversed, thus representing an obstacle. Additionally, an entrance cell and an exit cell are provided as input. The objective of the task is to determine a viable trajectory from the entrance cell to the exit cell.

Given a current position P and a grid-world state G (which includes the entrance and exist cells), we defined the constants $move_{up}$, $move_{down}$, $move_{left}$, and $move_{right}$. The motion policy was given by:

$$x = \operatorname{argmin}_i |dis_M(move_i, P \odot G)|.$$

For training, we employed a random walk algorithm, initiating from the entrance cell. If the walk reached the output cell within a maximum move limit, we utilized the random walk for training. Each position along each successful random walk became a training point for the algebra. We discarded walks that revisited the same cell. A distinct context constant was used for each walk.

An AML model was obtained as a result of this training, that was capable of finding reasonable paths between the entrance and exit cells, moving in straight lines when no obstacles impeded such movement. This outcome is particularly interesting, as the training set rarely contained any straight walks.

3.4 Experiments and Results

This section describes the experiments performed throughout the developed environments.

3.4.1 Bandit

The environment **Bandit-v0** implements an n -armed bandit problem. There is a single state (it is an effectively stateless environment), therefore the state/observation is not taken into account. Here, the **n_arms** parameter, which establishes the number of available actions, was set to 3. These possible actions are internally labelled by the AML engine as L1, L2 and L3. Regarding the reward, two factors must be taken into account in order to understand the approach taken to address this problem:

- At the time of writing, the AML engine has no explicit notion of a reward function, thus it must be fit in implicitly through the new problem formulation.
- At the time of running these experiments, only a discrete version of the AML engine was available.

Regarding the first point, and taking advantage of the stateless nature of the specific environment, the problem was reframed as a supervised learning problem where the objective is to effectively learn the reward associated to each action (as the Q-value only depends on the reward associated to the action). Secondly, while the original reward could have been discretized in “bins” (intervals) to address the second point, a much more simple approach was taken. Only three different rewards are possible, internally labelled by the AML engine as R1, R2 and R3. To enhance the comprehension of the results, reward correspondance was hard-coded such that action L1 in ground truth provides reward R1, action L2 in ground truth provides reward R2, and action L3 in ground truth provides reward R3.

Table 2: AML learning throughout batches on a simplified (deterministic) 3-armed Bandit problem.

Batch 1	R1	R2	R3
L1	1	1	1
L2	1	1	1
L3	1	1	0

Batch 2	R1	R2	R3
L1	1	1	1
L2	1	1	1
L3	1	1	0

Batch 10	R1	R2	R3
L1	0	1	1
L2	1	0	1
L3	1	1	0

In Table 2, the value of each matrix values determines the number of inconsistencies found by the AML engine. Therefore, a confusion matrix with a zero diagonal represents correct identification. The AML converges to this solution on its 10th batch of interactive learning, as depicted in the table.

3.4.2 Grid World

The environment **GridWorld-v0** implements a 2D grid world, previously depicted in Figure 8. Here, the state space size is determined by a map loaded from a comma separated value (.csv) file, which for the experiments is an 8x10 matrix. The actual used state is **Discrete** via the **BoxToDiscreteObservation** wrapper. The action space is fixed by the environment, composed by 8 actions which correspond to UP, RIGHT, DOWN, LEFT, and the intermediate combinations as in a conventional compass rose. The goal is fixed to represent a static environment. The starting point may change in order to enforce an agent policy that covers more states, as in exploration.

The same factors as those cited in subsection 3.4.1 are present. In this case, we cannot directly try to identify a map between the action and the reward, because the reward also depends on the state. Two solvers were implemented and tested:

- “Elite sample” method. On a conventional PC without optimizations, with uniform random initialization states, this method involved a mean of approximately 67 seconds to develop a policy that consistently reaches the goal.

- “Path planning” method. On a conventional PC without optimizations, with uniform random initialization states, this method involved a mean of less than 2 seconds to develop a policy that consistently reaches the goal.

The main conclusion that can be extracted from this experiment is the power of AML once the problem is explicitly presented to the algorithm via the correct embeddings.

3.4.3 Fake Ironing, TEO and TIAGo++ Ironing

From the agent’s perspective, the environments `FakeIroning-v0`, `TeoIroning-v0` and `TiagoIroning-v0` are all identical, and can be evaluated as a whole. In terms of implementation, they mostly differ on software dependencies (2D graphics for `FakeIroning-v0` and communications for `TeoIroning-v0` and `TiagoIroning-v0`) and may be merged in the future. Here, the state space size is determined by a map loaded from a comma separated value (.csv) file, which for the experiments is again an 8x10 matrix. Similarly, the actual used state is `Discrete` via the `BoxToDiscreteObservation` wrapper. Again, the action space is fixed by the environment, composed by 8 actions which correspond to UP, UP_RIGHT, RIGHT, DOWN_RIGHT, DOWN, DOWN_LEFT, LEFT, UP_LEFT). The starting point may change in order to enforce an agent policy that covers more states, as in exploration. The intermediate goals are wrinkles which are initially fixed to represent a static environment, and disappear as the corresponding state is covered by the agent. The episode finishes when all wrinkles disappear, as in the sequence of Figure 13, where the `TeoIroning-v0` is used in simulation.

On the other hand, `TiagoIroning-v0` was also tested in a real-world setting under the same conditions described above. Figure 14 illustrates this process through a motion sequence, depicting how the end-effector executes a series of UP_LEFT movements to remove a wrinkle. Additionally, Figure 15 presents a sequence of frames from the robot’s perspective, demonstrating the wrinkle’s removal done by the real TIAGO robot.

The same two methods as described in subsection 3.3 were applied for these environments. In both cases, solver times were linear in time with respect to the number of wrinkles:

- “Elite sample” method. On a conventional PC without optimizations, with uniform random initialization states, this method involved a mean of approximately 67 seconds to develop a policy that consistently reaches each wrinkle.
- “Path planning” method. On a conventional PC without optimizations, with uniform random initialization states, this method involved a mean of less than 2 seconds to develop a policy that consistently reaches each wrinkle.

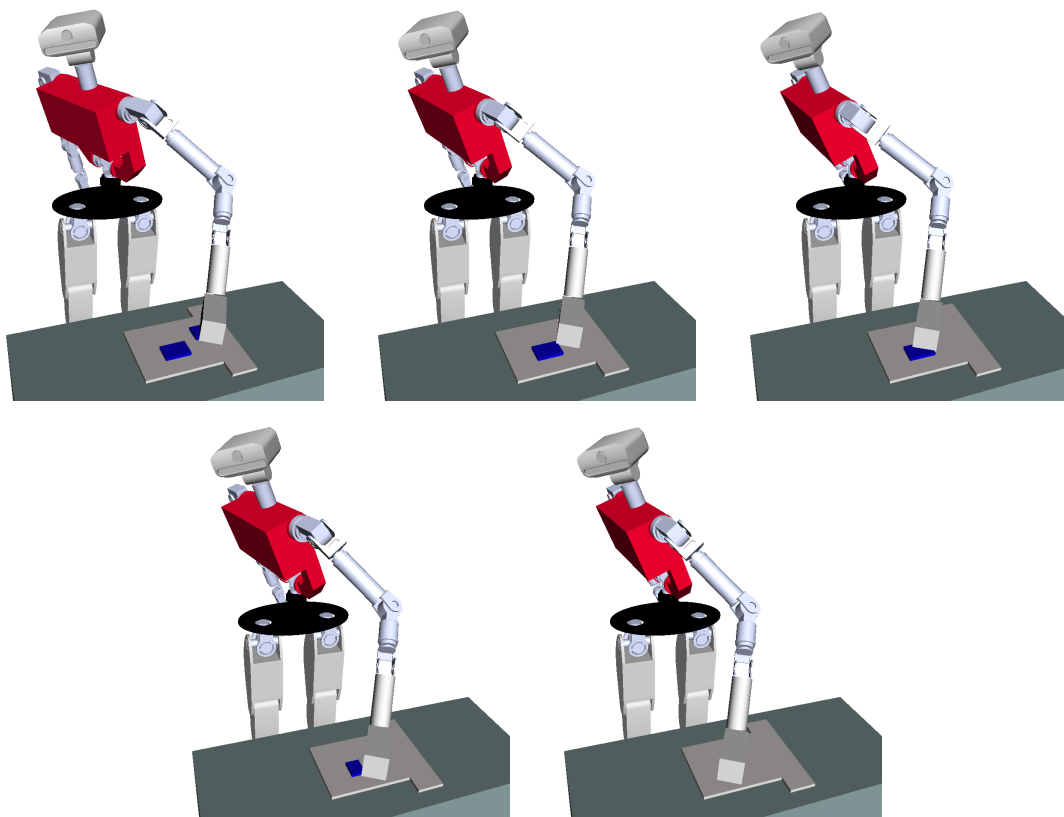


Figure 13: Completed sequence for 3 wrinkles, with TeoIroning-v0 connected to TEO humanoid simulator.

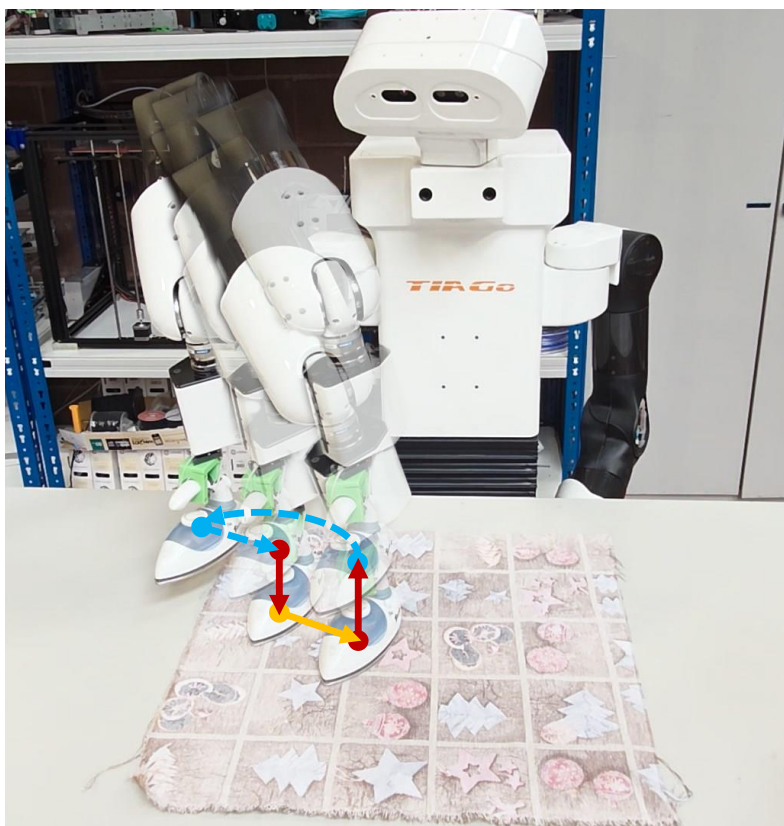


Figure 14: Motion picture of ironing task with real robot TIAGo++.

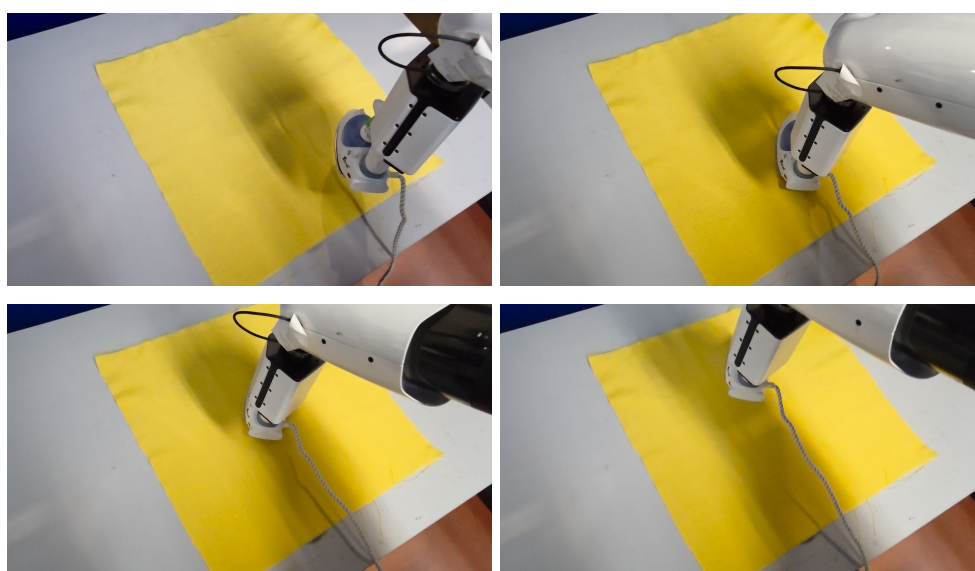


Figure 15: Completed sequence for a wrinkles, with *TiagoIroning-v0* connected to TIAGo++ humanoid real robot from its point of view.

4 Robotized Folding

Garments are deformable objects that can get wrinkled, and entangled with other garments. Therefore, it is difficult to recognize to which category a certain clothing article belongs with just a view of the wrinkled garment, even for a human. For that reason, the first step in our laundry pipeline is to extend the garment to recognize it more easily. This is what we call Unfolding, the step on which we focus within the broader definition of Robotized Folding.

In this sense, it can be divided into perception and manipulation.

- Perception: Algorithms and precision to detect heights of layers the garment surface. In this project, we leverage from previous developments [5] which provide the agent with an observation that has been transformed from a depth point cloud in color (via registration/alignment) in camera coordinates to a very simplified flat matrix, a.k.a the (rectified depth-map) image space for Robotized Folding.

Figure 16 illustrates the part of the pipeline of transition from full depth point cloud to the flat segmented garment, where height clusters are obtained via the watershed algorithm¹⁰.

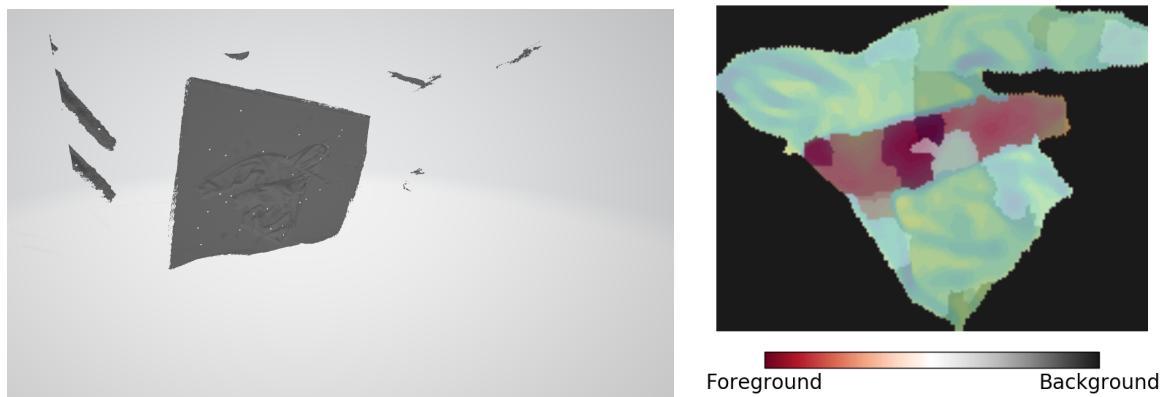


Figure 16: Transformation from full depth point cloud to the segmented garment (background removal) laying flat with obtained height clusters.

It is important to note that, with respect to the previous figure, only square/rectangular pieces of folded cloth have been used in the scope of this project, as those seen in Figure 7 and throughout the rest of this Section. We focused on this as a minimum viable demonstration to provide Supervised Learning with abundant samples.

All transformations are computed as part of the environments, namely (and with parameters specific to each camera and robot):

- From the pixels in image space to the coordinates in the $SE(3)$ space of the camera frame (via camera intrinsic parameters).
- From the coordinates in the $SE(3)$ space of the camera frame to the coordinates in the $SE(3)$ space of the robot root frame (via camera extrinsic parameters as well as robot joint positions and kinematic model).

¹⁰https://scikit-image.org/docs/0.24.x/auto_examples/segmentation/plot_watershed.html, accessed on March 19, 2025.

It is important to remark that the garment surface is provided to the agent as a flat discrete image, where each pixel corresponds to a discrete value that marks to which layer (merged similar height clusters) it belongs (lowest is without garment, first height involves one layer of cloth, second height involves two layers of cloth e.g. due to a fold).

Finally, an additional convenient mechanism introduced is that of polygonization, which can be obtained using the Ramer–Douglas–Peucker algorithm [11], see Figure 17. In the context of this project, this mechanism is not used to detect potential folds (as the AML approach is less ad-hoc than other methods which may require this information), but to simplify computations such as obtaining Rectangularity as a metric of error (under the above stated assumption of square/rectangular cloths).

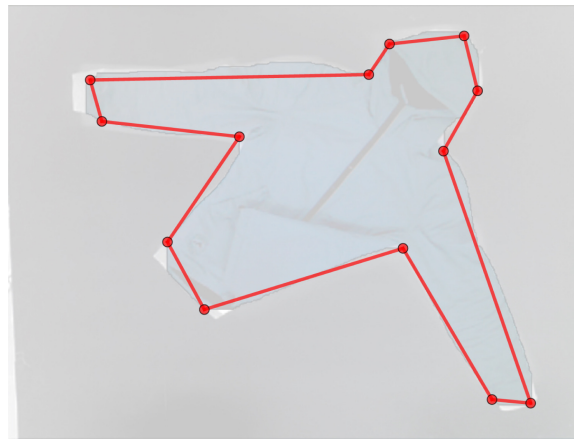


Figure 17: Polygonization of an arbitrary garment.

- **Manipulation:** Computation of trajectories based on perception. For all Robotized Folding environments, the action to command to the environment are simply four pixel values in image space: `pick_x`, `pick_y`, `place_x`, `place_y`. All transformations are computed as part of the environments, namely (in addition to the transformations which are also present in perception and applied here too for the actions which are in image space) from the targets in the $SE(3)$ space of the robot root frame to the actuators which are internally commanded in joint space (via the specific robot kinematics and forward/inverse kinematics and path generation for the robot end-effector).

. with this pair of values the robot's path planner is able to perform a generic pick and place task, that successfully result in the (un)folding operation.

While Robotized Ironing was performed via a Learning through Interaction paradigm, the problem statement for Robotized Folding was Supervised Learning. Specifically, the problem was framed as a Regression problem. Within this Regression problem statement, the labeled data was such that the rectified depth-map image corresponds to the input, and the pick and place coordinates correspond to the output.

4.1 Dataset and Ground-Truth Visualizer

A dataset of labeled data for Robotized Folding was created under the Supervised Learning paradigm, which has been openly published ¹¹. It generates images as those of Figure 18, as well as corresponding

¹¹<https://doi.org/10.5281/zenodo.14864393>

pick and place points in pixel coordinates.

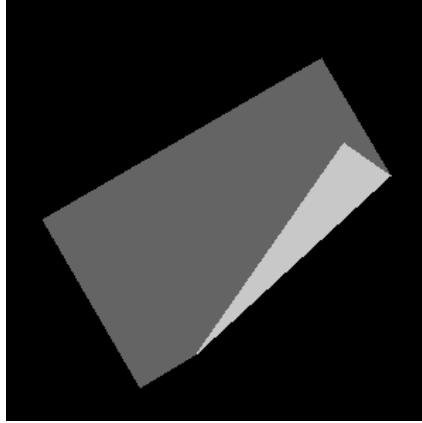


Figure 18: Representative image of the Robotized Folding dataset.

Dependency:

- pygame

Parameters:

- NUM_OUTPUT_SAMPLES = 10000
- IMAGE_SIZE = (300, 300)
- CLOTH_SIZE = (250, 150) # Before applying scaling (transformations are optional).
- CLOTH_TL = (25, 75) # TL is the top left corner before applying scaling/rotation (transformations are optional).
- RANDOM_SCALE_MAX = 0.25 # Scaling, shrinking random variation (do not apply expansion because it would go outside the image). Set to 0 for False.
- RANDOM_ROTATE = True # Rotation; if True, apply random 0-360 degrees.

Output:

- NUM_OUTPUT_SAMPLES images (grayscale emulating simplified 2.5D heights, rectangular fabric with a single fold that always starts from adjacent sides).
- A single 'labels.txt' file in plain text where each line is 'image_name.png, pick_x, pick_y, place_x, place_y'.

Additionally, a visualizer of this Ground-Truth data was developed. A screenshot of its functioning can be seen in Figure 19.

4.2 Environments

Two different environments have been developed within the scope of this task, each implemented as a Gymnasium¹² (previously OpenAI Gym) environment. Gymnasium is currently a de-facto standard which

¹²<https://github.com/Farama-Foundation/Gymnasium>, accessed on March 19, 2025.

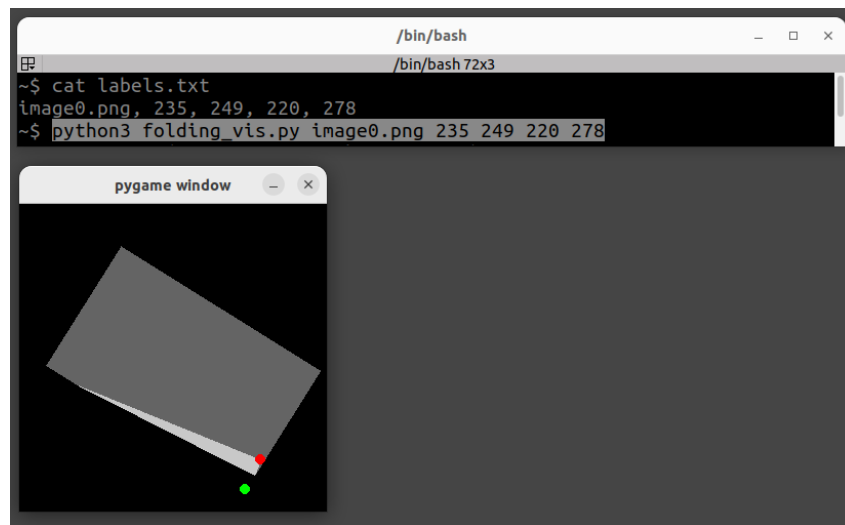


Figure 19: Ground-Truth Visualizer

includes simple pythonic interfaces that are capable of representing MDP, which is the general formalism used for representing Reinforcement Learning problems. It provides a thin layer of abstraction, such that the environments are cleanly decoupled from the algorithms in the code. This enables switching between environments while testing an algorithm, as well as testing different algorithms against the same environment.

These environments are generic and reusable in terms of solver algorithms. To enhance reusability of code and algorithms, they have all been designed to share common observation spaces as well as action spaces. Whereas the first two environments are popular concepts in the area of machine learning (specifically, among reinforcement learning practitioners), they have been reimplemented to “own” the code and enable seamless adaptation to any specific project requirement. The other two environments are variations on the robotic ironing task, which matches the work package objective of a domestic assistance robot task.

Finally, they have been released and open-sourced under a common repository: <https://github.com/roboticslab-uc3m/gymnasium-alma>

4.2.1 Folding: 2D (“Fake”)

The **FakeFolding-v0** environment provides testing against Ground Truth labeled pairs. Upon use, it loads a random image from the test dataset, as well as its corresponding ground truth from ‘labels.txt’.

Then, this environment is designed to evaluate the accuracy of the given pick and place pixels estimated by the solver algorithms. To this end, the observation of this environment is a Box of 100x100 float values within the bounds 0-255 to encode grayscale pixels of an image, while the action space is also a Box of four float values corresponding to the pixels coordinates of pick and place. At each step, the environment receives the estimated pick and place points and computes the quadratic error based on the Euclidean distance by comparing them to the known ground truth. This error measurement is then used as the reward value.

4.2.2 Folding: TIAGo++

The `TiagoFolding-v0` builds upon `FakeFolding-v0`, implementing a cloth folding task using the TIAGo++ robot. It can operate both to the simulation environment (see Figure 20) and with the real robot (see Figure 21).

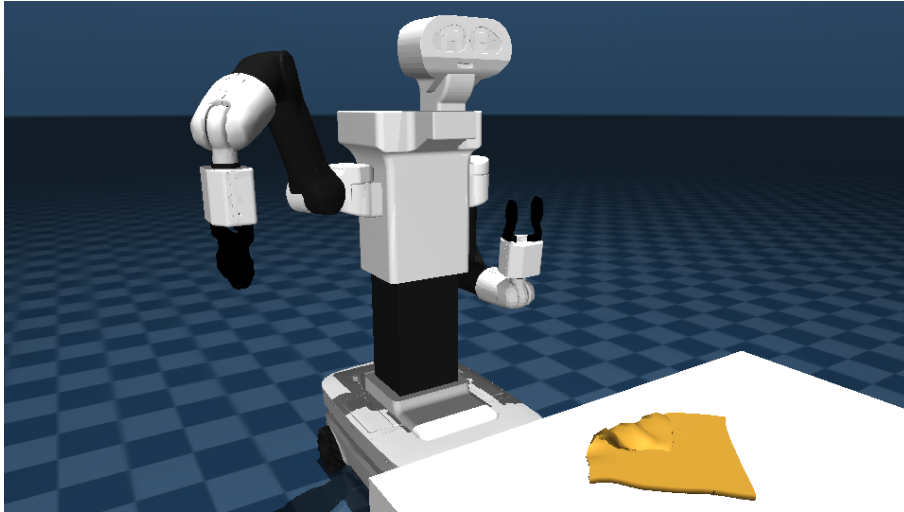


Figure 20: Screenshot of `TiagoFolding-v0` connected to TIAGo++ robot simulator.



Figure 21: `TiagoFolding-v0` connected to TIAGo++ real robot.

In the simulation setup, MuJoCo¹³ is used to render the TIAGo++ robot¹⁴, which interacts with a

¹³See <https://mujoco.readthedocs.io/en/stable/overview.html>, accessed on March 19, 2025.

¹⁴See https://github.com/pal-robotics-forks/mujoco_menagerie/tree/140ae8d30b430d9d8d8f0c42e031b93b59cb2968/pal_tiago_dual, accessed on March 19, 2025.

deformable cloth object. However, in both the simulation and real robot implementations, movements are executed via joint position commands communicated through ROS.

The observation space is defined as a Box of 100x100 representing grayscale pixel values in the range 0-255. The action space is also a box containing four values corresponding to the pick-and-place coordinates. Then, it initializes by taking a grayscale image of the cloth object using the camera of the TIAGo++ rectified and adapted as described at the beginning of this Section.

The step function receives the pick-and-place points detected from the image using the presented solvers and initiates the folding procedure. The first step involves determining the 3D positions of these points using depth information from the RGB-D camera and its intrinsic/extrinsic parameters and robot kinematics.

After transforming the image coordinates into 3D space, four key poses—pre-pick, pick, pre-place, and place—are computed using inverse kinematics. To achieve this, PyKDL, along with the TIAGo+ robot's kinematic model, enables the specification of end-effector poses. The positions of the pick and place poses correspond to the previously computed points, while the pre-pick and pre-place positions are set 0.15m above them. Then, the orientation for these poses are defined as follows:

- Roll: Set to π to ensure the gripper aperture remains horizontal.
- Pitch: Set to $\pi/2$ so that the fingertips make contact with the fabric.
- Yaw: Adjusted to align perpendicularly with the line connecting the pick and place points, ensuring the fabric corner is covered, as it is shown in the Figure 22.

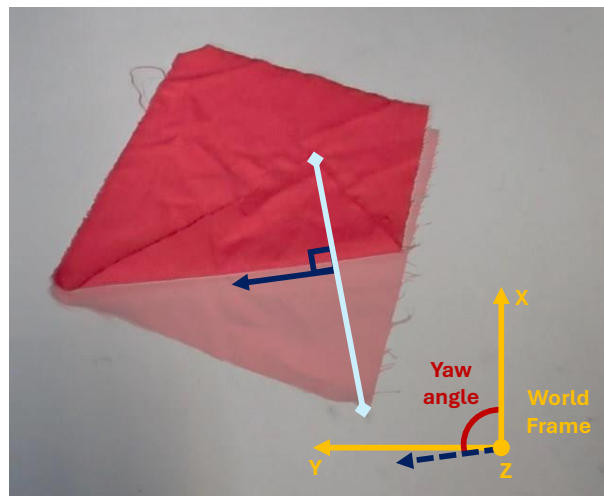


Figure 22: Yaw angle estimation for orientation in folding poses.

Since achieving the exact desired position and orientation may not always be feasible, a relaxation mechanism is applied. A tolerance of ± 10 degrees is considered for pitch and yaw angles. If the mentioned orientation is not feasible, inverse kinematics is computed for alternative angles within this range until four valid poses are found or the maximum number of attempts is reached.

Once the valid poses are determined, the robot follows these steps to execute the pick-and-place:

1. Moves to the pre-pick pose using the joint values obtained from inverse kinematics.
2. Executes a cartesian descent along the vertical axis until a variation in the vertical axis force is detected, indicating contact with the fabric.

3. Closes the gripper to grasp the fabric corner.
4. Returns to the pre-pick pose and transitions to the pre-place pose.
5. Performs another Cartesian descent until again a variation in the vertical axis force is detected, which means a contact with the placement surface.
6. Opens the gripper to release the fabric.
7. Returns to the pre-place pose and then to its initial configuration, completing the folding task.

This structured approach ensures precise execution of the folding procedure while maintaining adaptability to variations in the workspace (handled by a proper workspace calibration).

Finally, the reward is computed based on the error measurement using the concept of Rectangularity (feature-based learning as performed in [12]), calculated as the ratio between the area of the contour and the area of the minimum bounding box.

4.3 Solver Algorithms

Similar to how was performed for Robot Ironing, for Robot Folding, also two solver algorithms were tested. In this case, under the Supervised Learning paradigm (specifically, a regression problem), being the input an image, a Convolutional Neural Network was chosen, and the AML method was used as the main proposal of the project. To train each solver for robotic folding formulated as this regression problem (input: image, output: pick and place point), the code of the implemented dataset generator has been published at the repository <https://github.com/roboticslab-uc3m/alma-dataset>. Additionally, a dataset generated using this dataset generator has been uploaded to: <https://doi.org/10.5281/zenodo.14864392>

4.3.1 Baseline Algorithm: Convolutional Neural Network

As a baseline for supervised learning, we used a well-established Convolutional Neural Network (CNN) architecture. Establishing a widely recognized baseline allows us to assess the performance of the AML architecture by comparing its improvements against a robust reference model. The implementation code of this section is provided in the following repository: <https://github.com/roboticslab-uc3m/alma-dnn> that contains the baseline deep neural network for comparison on robotic folding.

The selected CNN structure, shown in Figure 23, has been widely applied to similar tasks, ensuring a reliable starting point while keeping computational requirements manageable. This baseline model provides a solid foundation for evaluating enhancements, serving as a benchmark against which alternative architectures can be measured. A detailed description of the network's layers and configurations is presented in Figure 24.

The CNN was implemented using Keras and trained with the Adam optimizer, minimizing a mean squared error (MSE) loss. The Adam optimizer was chosen for its adaptive learning rate capabilities, which help accelerate convergence and improve stability during training. The mean squared error loss was selected due to its effectiveness in regression-based tasks, aligning with the objectives of our supervised learning approach.

4.3.2 AML Methods: Regression

As stated in Section 3.3.2, at the beginning of the ALMA project, Algebraic Machine Learning (AML) [1], had demonstrated effectiveness primarily in supervised learning tasks, such as handwritten character recognition. In addition to what was commented in Section 3.3.2, later in the project, the engine was expanded to include the capability of handling real-valued data, both, as input and output, thus enabling

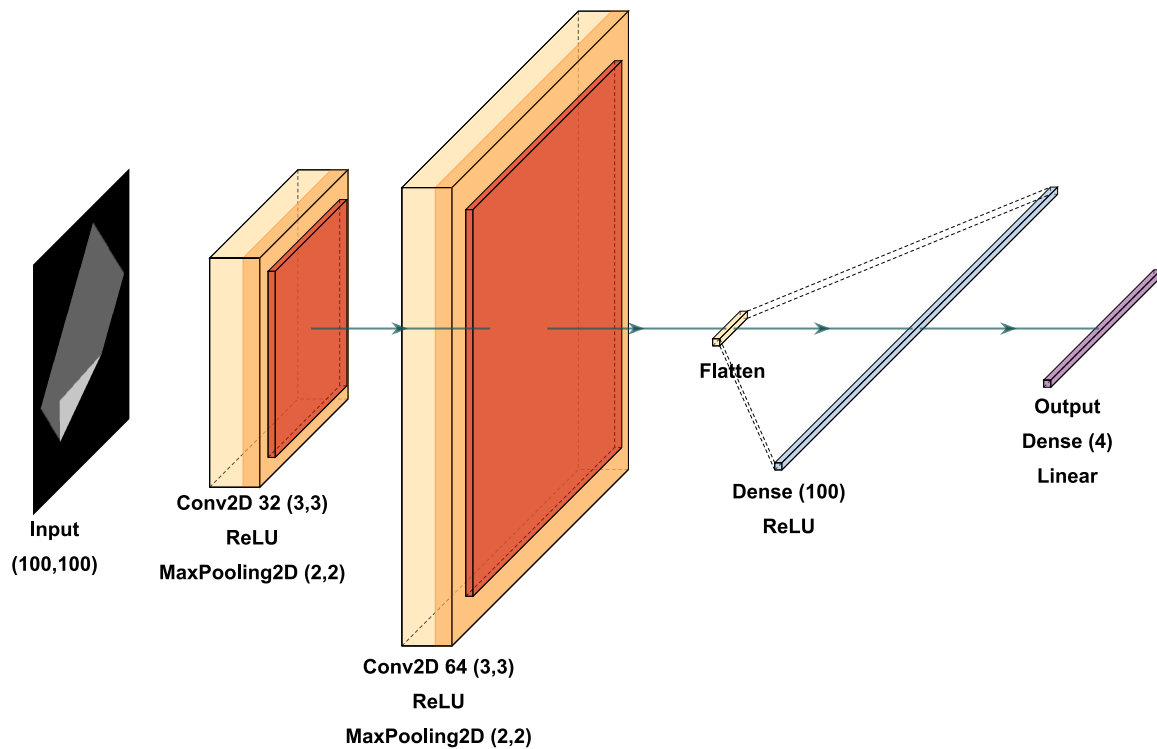


Figure 23: Convolutional Neural Network.

AML to solve regression problems. We employed regression to address the folding problem, albeit in a simplified manner, using cloth folding as an initial step towards the goal of solving clothing folding in a robotic scenario. This is the challenge we addressed, of using AML was determining the optimal pick-and-place coordinates for unfolding a sheet of cloth. The implementation code of this section is provided in the following repository: <https://github.com/roboticslab-uc3m/alma-aml> that contains the Algebraic Machine Learning (AML) applied algorithms.

The problem input consists of a 100×100 image, while the output comprises four numbers representing the two coordinates of the pick point and the two coordinates of the place point of the cloth. Each input image is a three-color representation of a rectangular sheet of cloth of varying sizes and proportions, with any angular orientation and folded along arbitrary lines. This gives a very large variability for the input images.

Given the high dimensionality of the images and the fact that no specific information was provided to the algebra to interpret images (no image bias), we consider this problem to be challenging. Additionally, this is not a classification problem but a regression problem with input in a space of 3^{10000} states and output in a four-dimensional space comprising 100^4 possible values.

The images were encoded using three constants for each of the 10,000 pixels. Each of the four output fields was modeled as the idempotent summation of a constant in an ascending chain and another constant in a descending chain. This approach has been used in various other ALMA deliverables, including human activity recognition and classifying images in datasets such as CIFAR-10 or MedMNIST. However, in this case, we were using the representation for output fields rather than input fields.

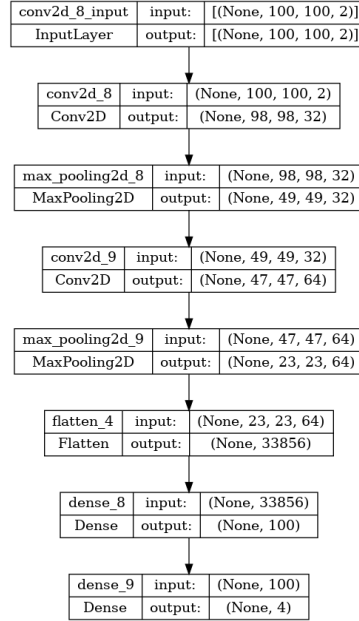


Figure 24: Convolutional Neural Network layer description.

According to this embedding method for real numbers, a value x is represented as:

$$g_x \odot l_x,$$

where l_x is a constant in an ascending chain (i.e., $l_x \leq l_{x+1}$) and g_x is a constant in a descending chain (i.e., $g_{x+1} \leq g_x$). Since there are 4 output fields, 8 chains are required.

Training was conducted in a supervised manner, and an algebraic model M was derived. The output for an image term I was computed as follows; for each of the four fields, we calculated:

$$x = \operatorname{argmin}_i |dis_M(g_i \odot l_i, I)|.$$

The algebraic model successfully located the pick and place positions for test images. The place position demonstrated high accuracy, typically within one pixel distance from the correct location. The pick position exhibited more uncertainty; typically, there existed a region for which the same discriminant size was obtained (as per the formula above), resulting in a square area a few pixels wide around the correct pick location.

4.4 Experiments and Results

This section describes the experiments performed with both **FakeFolding-v0** and **TiagoFolding-v0**.

4.4.1 Fake Folding

The experiments with this environment involved testing different images from the dataset, obtaining the pick and place points using both DNN and AML solvers, and evaluating the error using the **FakeFolding-v0** environment. In Table 3, the results obtained for three different images from the dataset are presented. Subsequently, the error between the ground truth and the solver results was calculated over 10 runs. The mean and standard deviation of these errors are shown in Table 4.

Table 3: Comparison of results of FakeFolding-v0 for three images from dataset

Image	Method	Pick X	Pick Y	Place X	Place Y
1000	Ground Truth	58	47	89	48
	DNN	54.94	45.32	86.95	48.10
	AML	54-67	37-53	90	52-54
		(misses 30)	(misses 16)	(misses 30)	(misses 16)
1001	Ground Truth	52	50	9	59
	DNN	52.02	46.12	7.90	54.85
	AML	44-47	43-48	34	75
		(misses 6)	(misses 0)	(misses 84)	(misses 104)
1004	Ground Truth	41	20	36	9
	DNN	37.37	16.57	34.05	9.79
	AML	44-45	52	53, 55, 56	84
		(misses 0)	(misses 3)	(misses 10)	(misses 52)

Table 4: Comparison of errors between Ground Truth and solver results for FakeFolding-v0 over 10 runs using dataset images. Mean and standard deviation (std) of the errors are calculated.

Method	Pick X Error	Pick Y Error	Place X Error	Place Y Error
DNN	3.1 ± 1.3	1.7 ± 0.8	2.2 ± 1.2	0.3 ± 0.9
AML	2.3 ± 1.4	4.1 ± 1.9	5.2 ± 2.0	8.2 ± 2.1
	(misses 1.2 ± 0.7)	(misses 0.9 ± 0.5)	(misses 1.5 ± 0.9)	(misses 2.4 ± 1.1)

4.4.2 TIAGo++ Folding

Experiments on folding with TIAGo++ were conducted in both simulation and real-world settings, with pick-and-place pixel points estimated using the ALMA algorithm.

In the simulation, the initial cloth configuration was provided as a grayscale image to the solvers, which estimated the pick-and-place pixel coordinates. These estimated points were then passed to the environment, which executed the folding task within the simulation, as shown in Figure 25. The resulting Rectangularity value over 10 runs was $22 \pm 6\%$, though it is important to note that the cloth simulation physics had a significant impact on the outcome.

The evaluation with the real-world robot followed the same procedure. The cloth image was provided to the solver in grayscale, and its output was passed to the environment for executing the folding task. This process is illustrated in Figure 26, where the resulting Rectangularity value over 10 runs was $97 \pm 2\%$.

A video containing all the simulations and experimental trials has been compiled and added as an additional material to this deliverable are published at our Youtube channel: @roboticaasistencialuc3m at UC3M

VIDEO LINK: ALMA D7.6 Uc3m Use cases

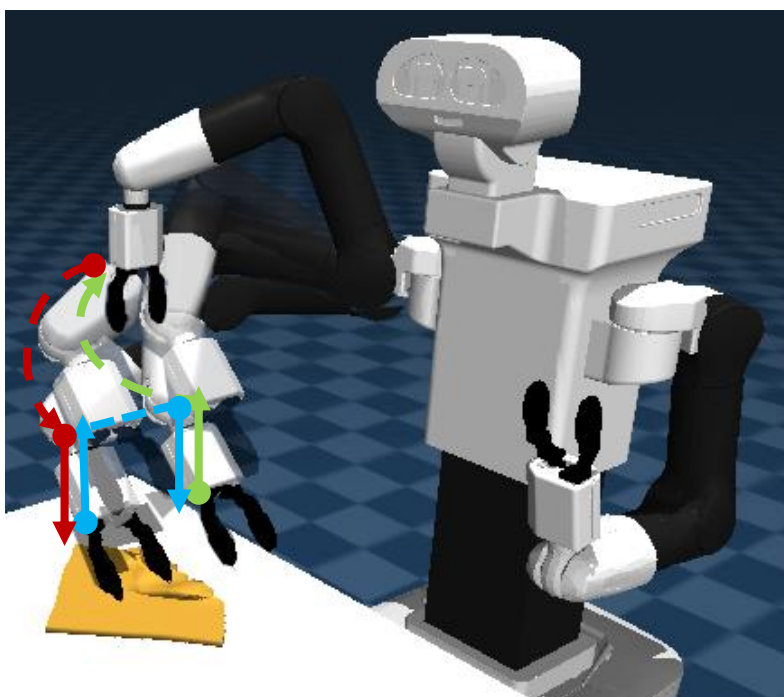


Figure 25: Motion picture of folding task with TiagoFolding-v0 and TIAGo++ simulation.

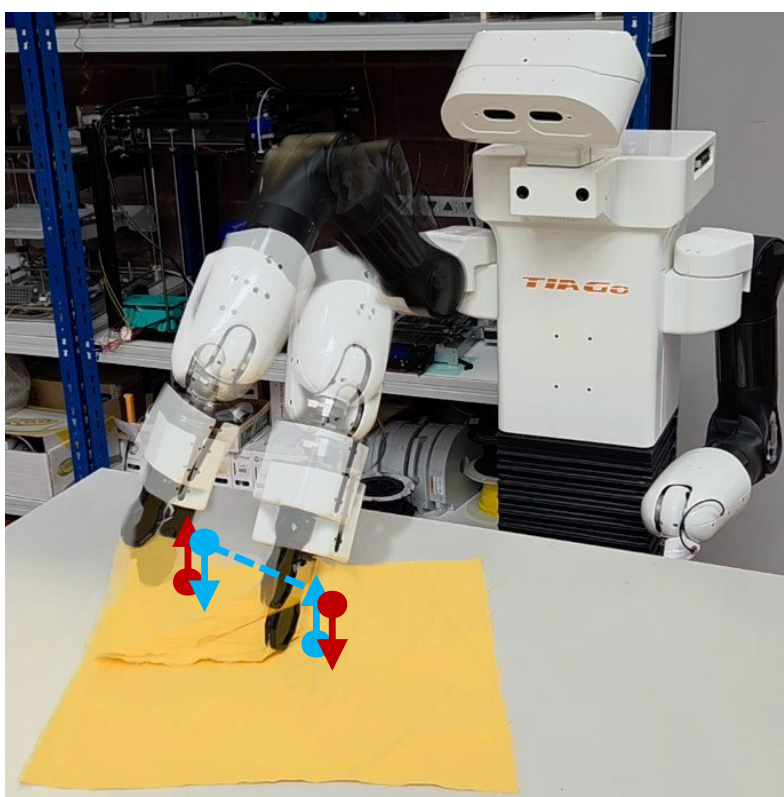


Figure 26: Motion picture of folding task with real robot TIAGo++.

5 Summary of Activities

This section lists the biweekly continuous development meetings, and includes a summary of the most relevant actions that led to the integration tasks described in the deliverable.

- (2020.12.16) Improved public installation guides on RGBD sensors. Further work on our object detection utilities and their interaction with existing applications. Introducing a surface reconstruction app using real depth frames in virtual environments.
- (2021.01.20) Migrating from OpenRAVE (YARP based simulator) to Gazebo for robot simulation. Webots has been also considered, yet Gazebo seems a better fit for our needs as it is reasonably lightweight and widely used in the robotics community, and due its ROS2 easy integration.
- (2021.02.03) Fixed an issue with low-level driver configuration regarding position/speed control. Investigating the improvement in smoothness of position-controlled trajectories. Considering an implementation of a cascading PID controller in Gazebo.
- (2021.02.17) Pending tests on real hardware: improved smoothness of position-controlled trajectories. Introducing an Orocos KDL-based solver for kinematic trees aimed for bimanipulation and locomotion tasks on TEO. Restoring continuous integration (CI) in our public repositories: migration from the Travis CI system to GitHub Actions.
- (2021.03.03) Spreading the adoption of GitHub Actions for continuous integration, also considering an automated generation of code documentation (Doxygen for API, GitBook for manuals and tutorials). Achieved a first functional Gazebo model of the TEO humanoid robot. Attempting a first approach to impedance control and compliance.
- (2021.03.31) Successfully verified we can perform high-frequency position-controlled trajectories. Started migrating code documentation to GitHub Pages (automated via GitHub Actions workflows). Fixed broadcasting of CAN (Controller Area Network) traffic via YARP ports for remote inspection and analysis.
- (2021.04.14) An upstream patch has been proposed and merged into YARP's codebase to enable absolute accuracy timers for use in periodically executed tasks. This aims to prevent time drifts and perform synchronization of CAN drives at precise intervals, specifically in scenarios that involve high-frequency position commands for time-critical applications.
- (2021.05.12) Finalized migration of continuous integration (CI) workflows to GitHub Actions alongside an upgrade of logging mechanisms. General review and refactorization of our repository of vision-related utilities. Following the conclusion of a long-running high-frequency position control issue, documentation has been enriched with tutorials and code examples.
- (2021.05.26) Resolved API deprecations following changes in YARP's upcoming stable release regarding frame grabber interfaces.
- (2021.06.23) Added a workaround in our kinematic tree solver to support closed-loop graph-like chains. We take advantage of this in bimanipulation scenarios where both TEO's arms (simple open-loop chains) are supposed to hold firmly an object, which means we can virtually close the loop and operate with a single TCP frame.
- (2021.07.07) We are preparing for the next minor release (3.5) of the YARP middleware. Our automated CI workflows had detected a blocker issue, which was promptly reported to YARP maintainers. We will proceed to bump minimum requirements in our software shortly after said release, further upgrades will follow. Initial considerations have been made regarding an implementation of friction compensation in TEO.

- (2021.07.21) Working on a CAN drop-in plugin that will support the widely-adopted SocketCAN Linux drivers. We have been using proprietary software from PEAK-System, yet future hardware enhancements on TEO might require a standardized approach.
- (2021.09.15) Finalizing migration from proprietary PEAK CAN drivers to mainline SocketCAN kernel drivers for Linux on TEO.
- (2021.10.06) Ongoing rework of TEO's head, including mechanical and software upgrades. A NVIDIA Jetson has been selected as the new CPU with GPU capabilities for intense operations oriented towards machine learning.
- (2021.10.27) CAN bus communications up and ready on the new head CPU to perform motor control. Testing gait algorithms on the real robot after succeeding in simulation.
- (2021.11.10) Ongoing work on a gravity compensation app involving force-torque sensors on the robot's arms.
- (2021.11.24) Our Apache Thrift IDL (Interface Description Language) files have been reviewed and extended. We use this for enabling simpler definitions and handling for RPC (Remote Procedure Call) services. Our current usage spans speech-related technologies (text-to-speech and automatic speech recognition) and specific demonstration-oriented applications.
- (2022.01.12)
 - Besides other additions in the recent YARP 3.6 release, we have aligned our low-level joint controller to comply with the new "IJointFault" C++ interface. It provides a means of exposing hardware faults (related to motor drivers) via API/GUI.
 - OpenCV 4.5.5 has been released. Now, a color-enabled KinectFusion algorithm (3D scene reconstruction) is available. Our wrapper has been updated to account for colored renders and point clouds.
 - A differential IK algorithm featured in the widely used KDL library has been integrated in our cartesian solver. It allows us to assign weights to specific joints or degrees of freedom in order to restrict their motion. This is especially suited for redundant mechanisms, but not only.
- (2022.02.02) An experimental hardware fix aimed at reducing the slack in lower-body joints has been introduced and tested on an offline-generated stable gait trajectory.
- (2022.02.16) Some electronics have been fixed or replaced involving TEO's grippers and the force-torque sensors on the wrists. We are currently testing the new hardware.
- (2022.03.02) Working on a motion capture (MOCAP) system for lower body limbs (previously was for torso and above), including Blender integration, performing fine calibration procedures.
- (2022.03.16) Preparing our platform for a new YARP feature: friction coefficients are now exposed for use by joint controller implementors.
- (2022.09.28) Successfully tested a trajectory teaching application on TEO using force-torque sensors, roughly mimicking classical gravity compensation (but in cartesian space). Currently working on a new joint controller, thus allowing impedance control and compliance, among other specifically torque-oriented applications.
- (2022.10.26) Revisiting previous work on TEO's gait controllers. We decided to start anew with an incremental approach, currently testing single-foot stability through compensation of ZMP (zero moment point). To properly test our algorithms, the simulated Gazebo model gained the previously missing FT (force-torque) and inertial (IMU) sensors, replicating what real TEO has aboard. The

new joint controller is partially ready, now awaiting measurements and estimation of static friction and PID coefficients.

- (2022.11.23) More work on single-foot stability on TEO. Mechanical failures have been encountered and eventually fixed, we are still working on a means to compensate for excessive slack using FT/IMU sensors.
- (2022.12.07) Advances in gait generation and dual arm movement coordination. Working on the new head's electronics. Revisiting the F/T sensor software (possibly with future modifications in the acquisition hardware).
- (2023.01.18) Finished the first implementation of a custom PID controller overriding the one we have been using, which is embedded in the proprietary firmware. This could enable us to explore previously unreachable features (e.g. impedance control). Now working on the migration of the acquisition hardware related to force-torque sensors. Currently using a closed-source and closed-hardware solution, ideally we would want to keep it as open as possible.
- (2023.02.15) Reworked our offline gait trajectory generator, next we are aiming to compensate for slack through offline estimations and online data using an inertial sensor. Improved behavior upon reaching joint limits. Tested a new gravity compensation and impedance control app using the force-torque sensors to detect external forces.
- (2023.03.01) Fixed an issue in low-level communications: to avoid message bloat, CAN writes will be paused as soon as we detect that the drives are not responding.
- (2023.03.15) Continuing work on a new firmware for reading from the force-torque sensors. Investigating whole-body controllers regarding robot stability. Tests on reliability of internal CAN-Open comms mechanisms were delivered.
- (2023.03.29) Fixed an upstream change that broke our plugins for the OpenRAVE simulator. Also identified an issue with the PID tuning on our Gazebo models.
- (2023.04.12) Aiming for the upcoming hardware refurbishment of TEO's head, we are implementing a YARP-compatible device to communicate with a GenICam USB/RGB camera that will complement the Intel RealSense depth cameras currently in use.
- (2023.06.21) Hardware fixes on TEO's arms and leg. Updates in code for sensorial pipeline processing.
- (2023.09.13) Tested the new open-source firmware for the force-torque sensors. Prepared a draft implementation for a software layer wrapped by YARP to interface with said FT sensors and expose their readings over the network.
- (2023.09.27) The software layer for capturing and exposing data acquired from the force-torque sensors is almost ready. All Cartesian-space level commands tested and available from Python, and additionally developed C++ simulator plugin for 3D visualization of ironing task progress.
- (2023.10.11) Small post-experiments code cleanup. Pending actual refactoring and comparisons.
- (2023.10.25) Started actual refactoring following the post-experiments code cleanup.
- (2023.11.22) Software integration of ROS2 (Humble) on TEO, hardware improvements on its power supply.
- (2024.01.17) Following research on soft object simulation (towards folding) after the holiday break. Integrated force sensor data acquisition fully tested and working well.
- (2024.02.14) Further research on deformable object simulation (towards folding) with special interest on differentiable physical simulation.

- (2024.02.28) Hardware upgrades in process on the actual physical humanoid robot: mechanics (mostly head) and electronics (on head, but also central components). Check of reach and fine positioning capabilities of TEO and TIAGO robot's arm.
- (2024.03.13) Still working on hardware platform improvements (head and trunk components) and software at simulation level. Glue code and testing reliability of comms capabilities YARP/ROS2.
- (2024.04.10) More research on deformable object simulation. Also more basic simulation, the process is leading to interest in the integrator part of simulators.
- (2024.05.08) Force/torque sensor acquisition hardware: completion of functional prototype of hardware (replaces old PCI board). Also working with AML engine searching for semantic embedding that fits paper with VTT.
- (2024.06.19) Progress with AML toy examples towards robotic garment folding. Reviewing state of the art papers and software garment folding. Article published (DOI: 10.3390/informatics11020036) with VTT.
- (2024.07.17) Reviewing state of the art papers and software garment folding.
- (2024.09.11) Managed to run latest version of AML engine on Docker (with correct Python and OpenSSL version). Initial thoughts on most desirable framework for learning (with actual inference).
- (2024.10.09) Progress continues regarding simulation of flexible materials towards training for robotic folding/unfolding.
- (2024.12.04) Development of synthetic data generator of labeled grayscale images for robotic unfolding. Scales to generating 10k samples in 1–2 min.

6 Summary of related code repositories and dataset

In this section we just summarize the data related to the code repositories created along the project and some results, in terms of code, publications and video materials.

Public Repositories

Algebraic Machine Learning (AML) applied algorithms:

<https://github.com/roboticslab-uc3m/alma-aml>

Robotic environments for applying AML algorithms:

<https://github.com/roboticslab-uc3m/gymnasium-alma>

Baseline deep neural network for comparison on robotic folding:

<https://github.com/roboticslab-uc3m/alma-dnn>

Dataset generator to train AML algorithm for robotic folding:

<https://github.com/roboticslab-uc3m/alma-dataset>

Generated dataset: <https://doi.org/10.5281/zenodo.14864392>

Complementary video materials: published at our Youtube channel: @roboticaasistencialuc3m at UC3M

VIDEO LINK: ALMA D7.6 Uc3m Use cases

Publications

- Stephen Fox and Juan G. Victores. Safety of human–artificial intelligence systems: Applying safety science to analyze loopholes in interactions between human organizations, artificial intelligence, and individual people. Informatics, 2024. doi: 10.3390/informatics11020036. URL Online (Q2)
- Juan G. Victores et al. Algebraic Machine Learning for Robotic Garment Folding/Unfolding. Simposio CEA de Robótica, Bioingeniería, Visión Artificial y Automática Marina, 2025. (Upcoming)

References

- [1] Fernando Martin-Maroto and Gonzalo G. de Polavieja. *Algebraic Machine Learning*. 2018. arXiv: 1803.05252.
- [2] Santiago Martínez et al. “Teo: Full-size humanoid robot design powered by a fuel cell system”. In: *Cybernetics and Systems* 43.3 (2012), pp. 163–180. DOI: 10.1080/01969722.2012.659977.
- [3] Rosen Diankov and James Kuffner. “Openrave: A planning architecture for autonomous robotics”. In: *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34* (2008).
- [4] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. “YARP: yet another robot platform”. In: *International Journal of Advanced Robotic Systems* 3.1 (2006), p. 8.
- [5] David Estevez et al. “Enabling garment-agnostic laundry tasks for a Robot Household Companion”. In: *Robotics and Autonomous Systems* 123 (2020), p. 103330. ISSN: 0921-8890. DOI: <https://doi.org/10.1016/j.robot.2019.103330>.
- [6] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more*. Packt Publishing Ltd, 2020.
- [7] Pieter-Tjerk De Boer et al. “A tutorial on the cross-entropy method”. In: *Annals of operations research* 134 (2005), pp. 19–67.
- [8] Grady Williams et al. “Information theoretic MPC for model-based reinforcement learning”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1714–1721.
- [9] Fernando Martin-Maroto and Gonzalo G de Polavieja. *Semantic Embeddings in Semilattices*. 2022. arXiv: 2205.12618.
- [10] Fernando Martin-Maroto and Gonzalo G de Polavieja. *Finite Atomized Semilattices*. May 2021. arXiv: 2102.08050.
- [11] David H Douglas and Thomas K Peucker. “Algorithms for the reduction of the number of points required to represent a digitized line or its caricature”. In: *Cartographica: the international journal for geographic information and geovisualization* 10.2 (1973), pp. 112–122.
- [12] Juan G. Victores. “Robot Imagination System”. PhD thesis. Universidad Carlos III de Madrid, 2014. URL: <https://hdl.handle.net/10016/19834>.