Simulation of an autonomous and adaptive robotic arm

Juan Diego Lozada

Cristian Santiago López Cadena

Report

Universidad Distrital Francisco Jose de Caldas

System Sciences Foundation

Carlos Andrés Sierra

09 de Abril del 2025

Systems Design Framework

Functional Requirements

- The arm must detect the objects it must transport in its environment using computer vision.
- It must classify detected objects by type and physical characteristics such as shape and material.
- It must be able to calculate the optimal grasping point based on the object's geometry, dimensions, and orientation.
- Calculate the optimal trajectory to the grasping point and constantly update this trajectory based on the object's movement and the arm's range of action.
- The arm must calculate how much force to exert to lift the object without damaging it.
- It must calculate the optimal trajectory, without collisions, to carry the object to the delivery point and release it without causing damage.
- It must be able to report if a task cannot be performed.
- It must learn optimal strategies that allow it to be more efficient, correct errors, and adapt to changes in its environment in real time.

Non-Functional Requirements

- The robotic arm must achieve and maintain a minimum accuracy of 80
- It must have a response time to changes in the environment of less than 1 second
- The architecture should be modular, so that components such as vision or arm range can be updated independently.

- The system must present logs and reports with metrics to monitor and evaluate the learning process.

Components

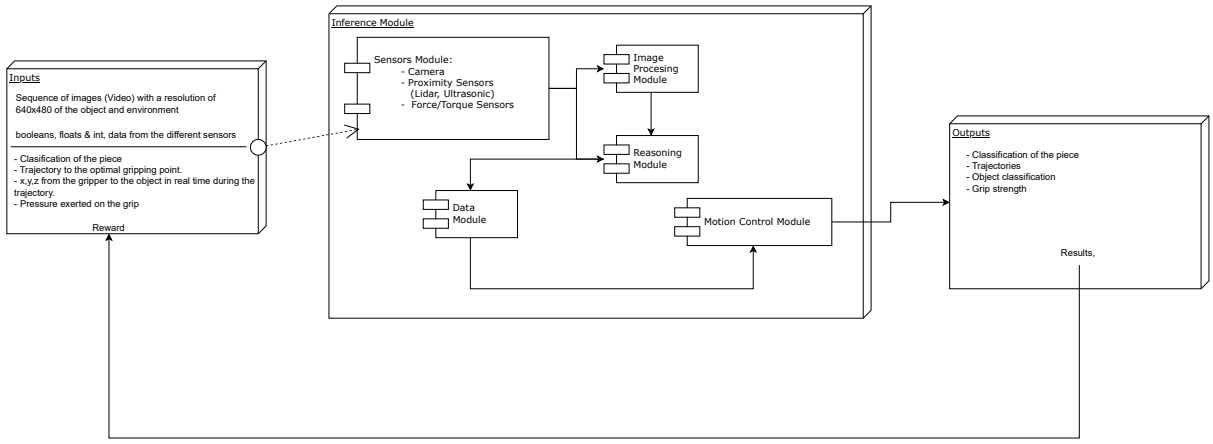


Figure 1

Component diagram

Sensor Module.

- Camera: Detect the object and analyze the shape of it
- Force/Torque Sensors: Pure Force, measure the pression made by the fingers, avoiding break the object; Pure torque Sensors, enable a sense of the torque being applied or experienced by a section of a robotic arm; Force/Torque (FT)
- Proximity Sensor: Calculate distance between the arm and the object

Image Processing Module.

- Object Detection: Use OpenCV to identify objects could appear inside the area of the camera, through contours and segmentation
- Object Classification: Implement AI models (TensorFlow) to recognize object types and determine optimal gripping strategies.

- Object Position: Calculate real-time object coordinates to optimize the robotic arm's approach and gripping precision.

Reasoning module.

- Reinforcement Learning: Train the robotic arm using reward-based feedback to improve object gripping efficiency over time.
- Environment Simulation: Use Gym/OpenAI Gym to train the robotic arm in a controlled, physics-based virtual environment.

Motion Control Module .

- Trayectory generation: Adjust paths without collision using reinforcement learning
- Grip Strenght & angle adjustment: Adjust the strenght depend on the object and the angle with the correct way to grab, and use reinforcement learning to improve the process

Data Module.

- Sensor Data Processing: Captures and filters data from the camera, force, and proximity sensors
- Grip Force Analysis: Records applied pressure to optimize gripping

Feedback Loop

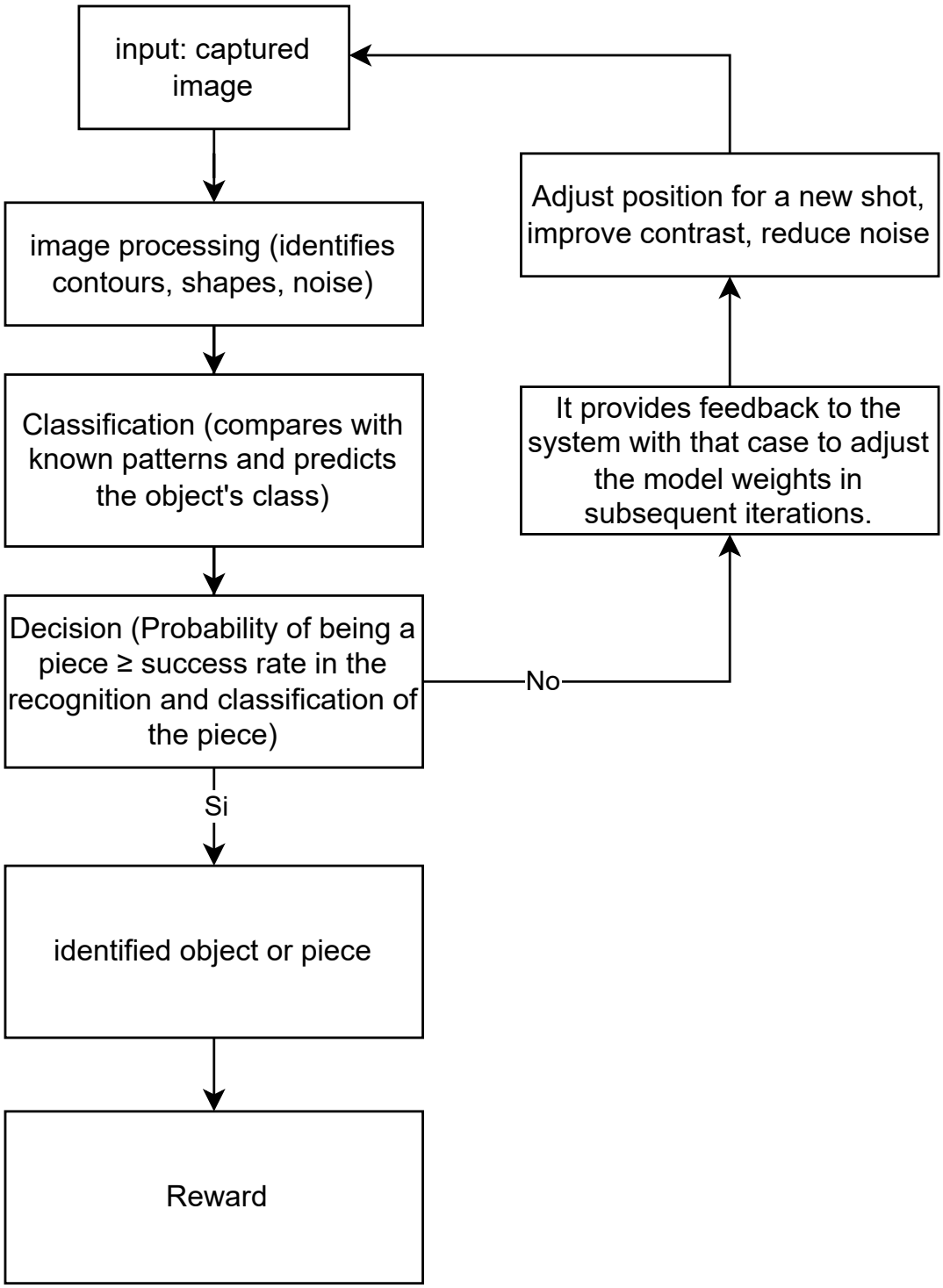


Figure 2

feedback loop identify object/part

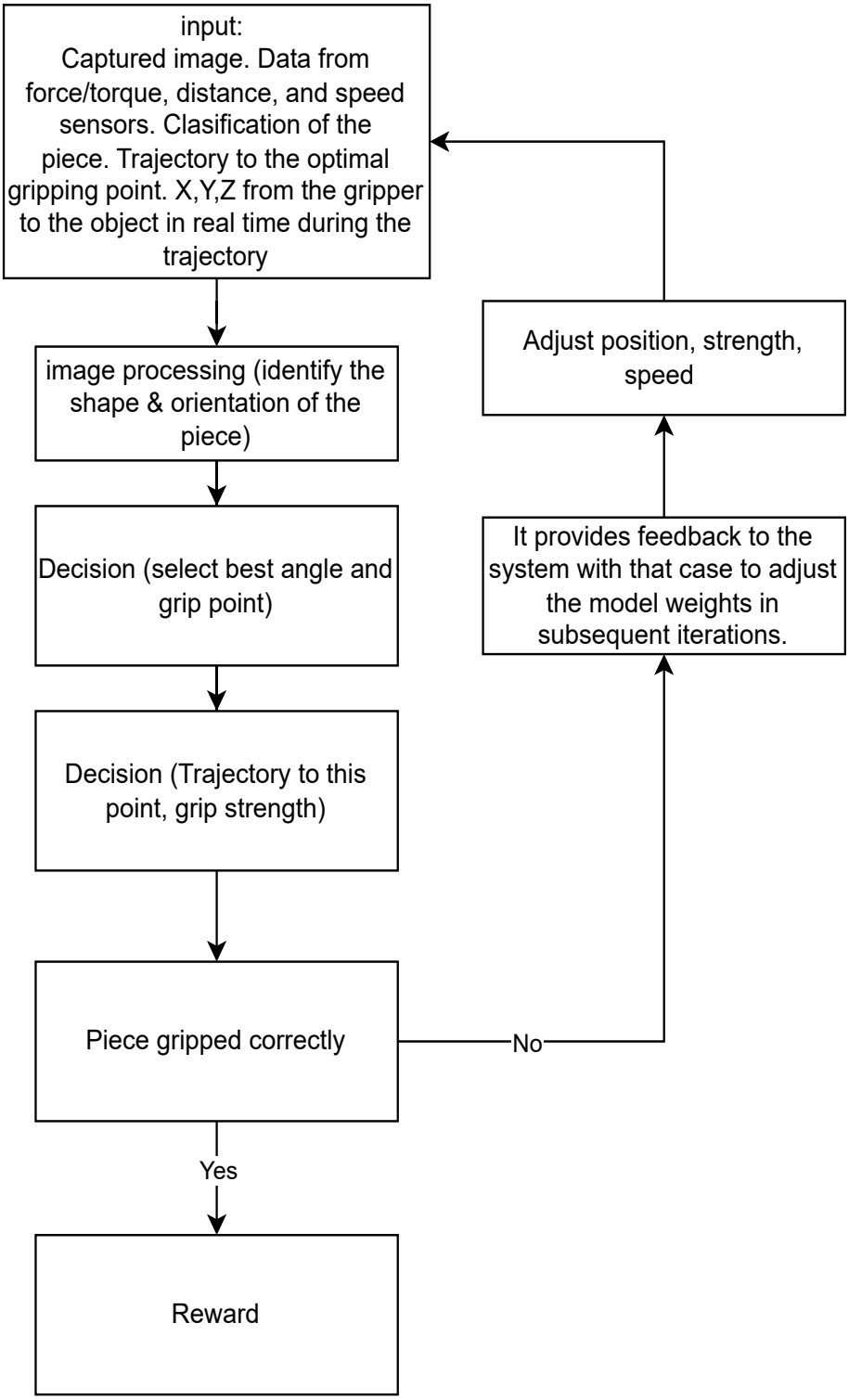


Figure 3

feedback loop grab piece/object

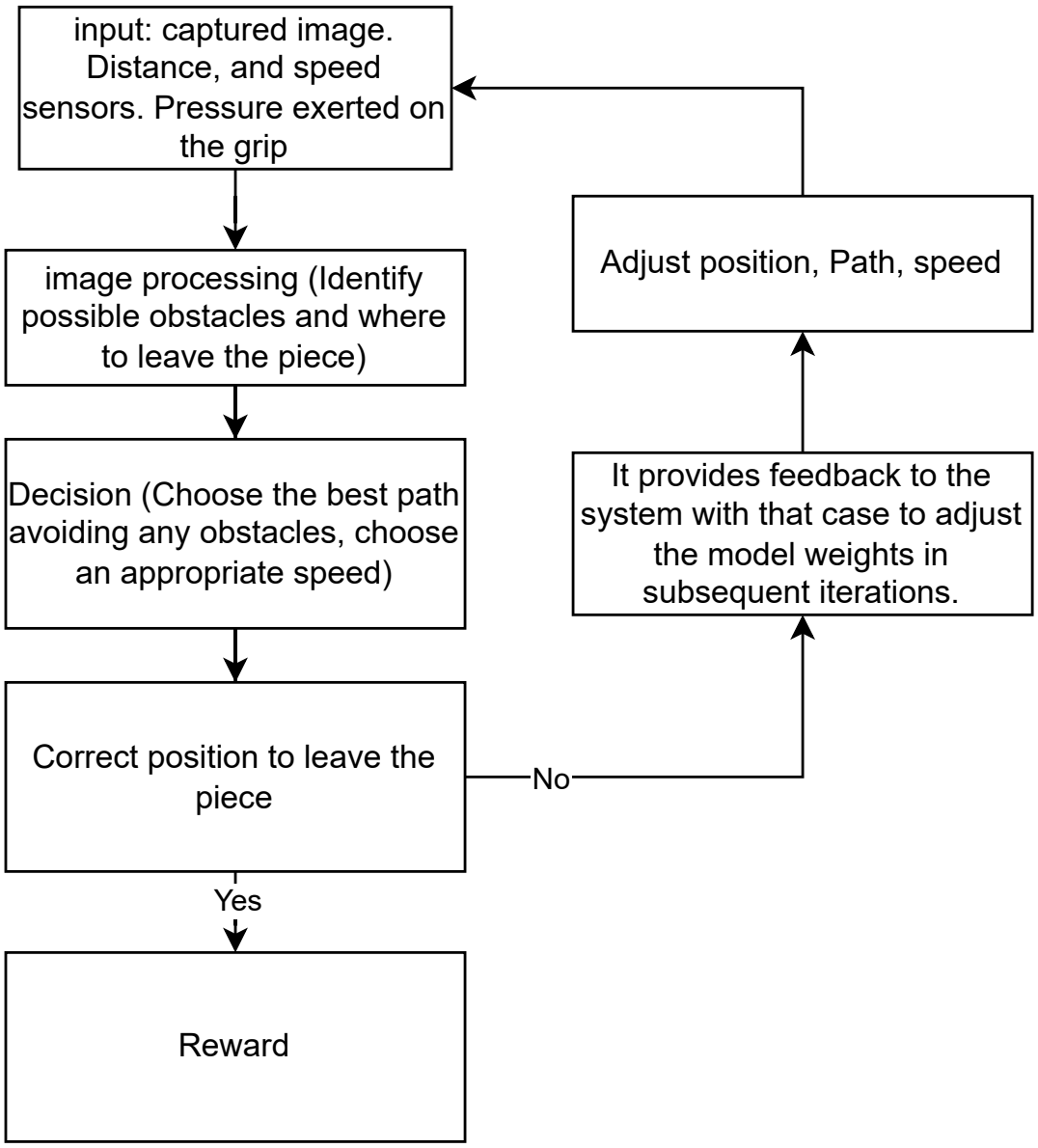


Figure 4

feedback loop transporting part/object

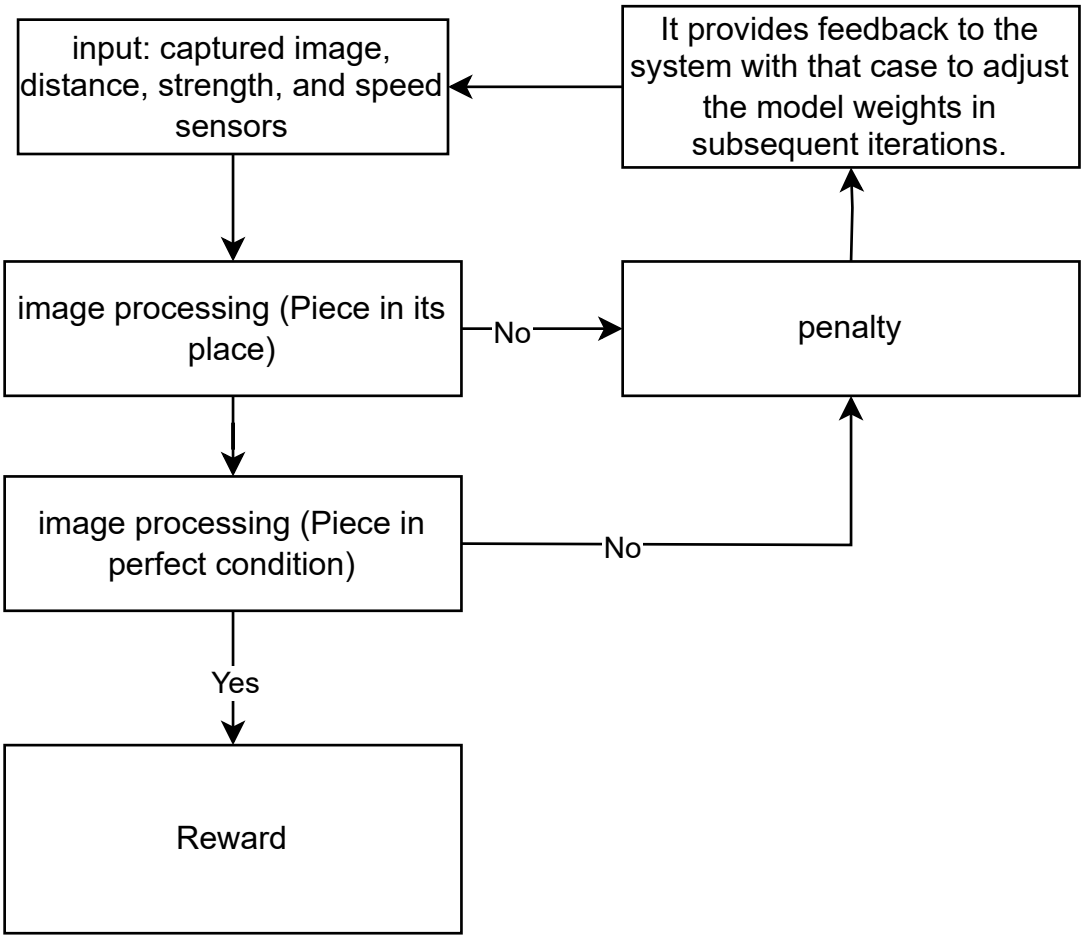


Figure 5

feedback loop drop piece/object

- Reinforcement Reward Loop: Assigns rewards based on successful gripping and movement efficiency
- Trajectory Correction Loop: Adjusts the path in real-time to avoid errors or obstacles
- Vision-Based Feedback Loop: Recalculates object position if it moves unexpectedly

Potential algorithm and frameworks

For object detection and classification, we recommend using a convolutional neural network, which is a deep neural network used for computer vision tasks such as those mentioned above. This is because convolutional networks are especially effective at extracting spatial and hierarchical features such as edges, shapes, textures, or complex patterns from images using convolutional filters. To implement this network, we will use the TensorFlow framework, more specifically its object detection API, which allows us to train, evaluate, and implement object detection models efficiently using pre-trained models. Use a pre-trained model will help us to reach the goal to have a sucess rate elevated

System Dynamics Analysis

The system's operation is structured around four fundamental operational phases, which the robotic arm moves through sequentially and with feedback:

1. Environment and object recognition: The agent receives a structured observation vector from the simulator. This includes key parameters such as the 3D position and orientation of the target object, the joint positions of the robotic arm, the state of the gripper (open or closed), and the distance between the gripper and the object. These numerical inputs form the basis of the agent's understanding of the environment.

2. Grasp planning and execution: Once the part has been detected and classified, the system estimates the optimal grasping point based on its geometry and center of mass. An initial approach trajectory is generated, which is dynamically updated through sensory feedback (vision and position) to compensate for object displacement. Upon reaching the action zone, the grip is performed, adjusting the force and closing angle according to the pressure sensors.

3. Part transport: The arm calculates the optimal route to the placement destination. During this journey, the force exerted on the object is monitored in real time, ensuring it remains within safe thresholds to prevent damage or falls, and adjustments are made as variations are detected.

4. Release and verification: Upon reaching the delivery point, the system executes the object release sequence. A return trajectory to the resting point is then planned, but first, it positions itself to capture an image of the deposited object. This image is used as feedback to verify whether the part was placed correctly and in optimal conditions, thus informing the agent's learning process for future tasks.

Phase diagrams

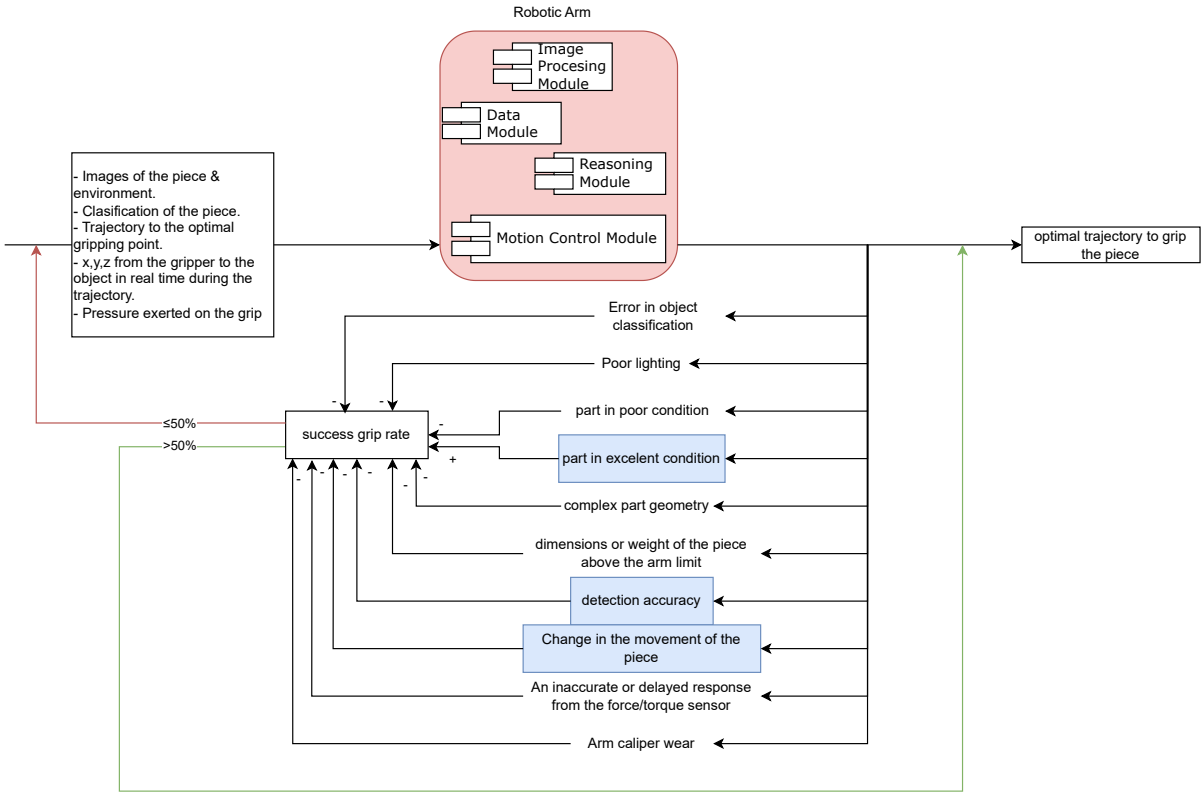


Figure 6

Grasp planning and execution phase

As evidenced in the phase diagram presented above (Figure 6), multiple factors have been identified that can negatively and positively affect the success rates associated with the phase Grasp planning and execution phase. These factors include:

- Error in object classification: If the part was classified incorrectly in the previous

phase, a correct gripping point cannot be calculated, preventing the system from completing its objective.

- Poor lighting: Since the system interacts with the environment via video, if lighting conditions are not ideal, the environment could be misdetected, the part could be missed, or its position could be misdetected.
- Dimensions or weight of the part above the arm limit: If the dimensions or weight of the object exceed the arm's capabilities, the arm could suffer physical damage, preventing it from not only fulfilling the phase's objective but also its overall objective.
- Change in the movement of the piece: If the piece remains in constant motion, it would make it difficult for the arm to approach and would force it to process more information and constantly update itself, which generates greater wear and processing costs.

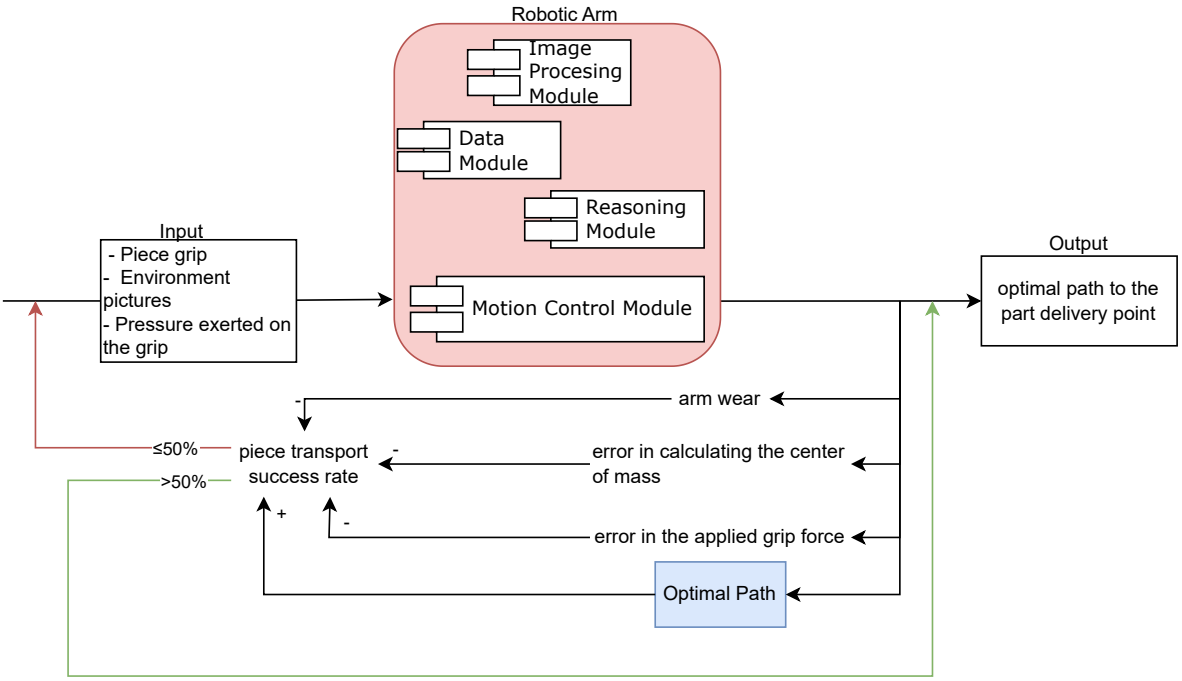


Figure 7

Part transport phase

As evidenced in the phase diagram presented above (Figure 7), multiple factors have been identified that can affect, both negatively and positively, the success rates associated with the part transport phase. These factors include:

- Arm wear: This is a physical factor that can affect part transport. If the arm is not in optimal condition, it can cause mid-process failures, such as the part coming loose or disassembling in the middle of the process, or it can also affect the transport speed, etc.
- Error in calculating the center of mass: If the center of mass is calculated incorrectly, the risk of the object falling during transport or being damaged increases.
- Error in the applied grip force: If more force than necessary is applied, the part could be damaged, and if too little force is applied, the object could fall and be damaged.

- Optimal path: If the calculated trajectory is optimal, transport can be carried out more safely and efficiently.

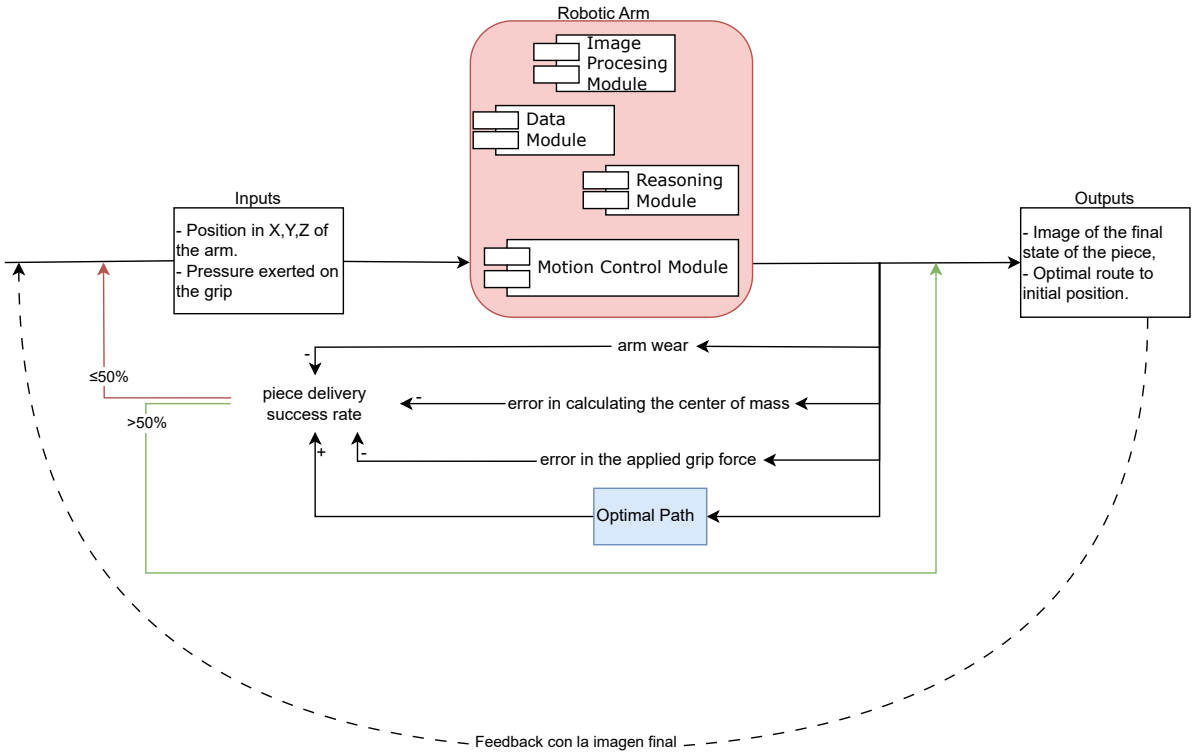


Figure 8

Release and verification phase

As evidenced in the phase diagram presented above (Figure 8), multiple factors have been identified that can affect, both negatively and positively, the success rates associated with the Release and Verification phase. These factors include:

- Arm wear: This is a physical factor that can affect the transport of the part. If the arm is not in optimal condition, it can fail mid-release, releasing the part or simply stopping working, resulting in the part not being delivered.

- Error in calculating the center of mass: If the center of mass is calculated incorrectly, the risk of the object falling during transport or being damaged increases.

- Error in the applied grip force: If more force than necessary is applied, the part could be damaged, and if too little force is applied, the object could fall and be damaged.
- Optimal path: if the calculated route is optimal, the transport can be carried out with greater safety and efficiency, towards the delivery point.

In this phase, the key lies in the general feedback from the system when delivering the part.

The following performance metrics were also defined:

1. Piece grip success rate (Phase 1, Figura 1).
2. Piece transport success rate (Phase 2, Figure 2).
3. Piece delivery success rate (Phase 3, Figura 3).

For each phase, performance thresholds were established that allow for adaptive decisions. If the success rate is less than or equal to 50%, the system activates feedback mechanisms to reanalyze inputs and collect new data. If it exceeds this threshold, the system continues executing the current phase's objective. These initial thresholds are considered provisional and will be progressively optimized using reinforce learning techniques, aiming to achieve a success rate equal to or greater than 80%.

Mathematical/Simulation Model

To mathematically describe the behavior of a robotic arm, we focus on the spatial configuration of the arm's actuation. For this, we resort to the direct kinematic model using homogeneous transformation matrices, supported by the Denavit-Hartenberg (D-H) convention. This formulation allows us to represent the position and orientation of the arm's gripper as a function of its joint parameters. This is thanks to the fact that direct kinematics determines the position and orientation of the robotic arm's end effector with respect to a base coordinate system, given a specific joint configuration. For an arm with n links, each link is represented by a homogeneous transformation matrix $T_i \in R^{4\times4}$ that

relates the link's coordinate system i with that of the link $i-1$.

$$T_0^n = T_1 \times T_2 \times ... \times T_n$$

(1)

Each transformation T_i is defined by the four D-H parameters:

- θ_i: angle of rotation around the axis z_{i-1}
- d_i: displacement along z_{i-1}
- a_i: link length projected onto the axis x_i
- α_i:angle between z_{i-1} y z_i, measured around x_i

Then the homogeneous matrix associated with each link is:

$$T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

On the other hand, the position and orientation of the object is represented by a homogeneous matrix $T_{objeto} \in SE$ which can be obtained using computer vision techniques such as PnP or Extrinsic Calibration.

$$T_{\text{objeto}} = \begin{bmatrix} R_{\text{obj}} & t_{\text{obj}} \\ 000 & 1 \end{bmatrix} \quad (3)$$

Where

- $R_{obj} \in R^{3\times3}$ is the orientation of the object.
- $t_{obj} = [xyz]^T$ is its position relative to the robot's base system.

Finally, we proceed to plan and control the movement. To do this, once the joint configuration necessary to reach the desired point has been determined, a smooth trajectory is generated between the initial and final positions in the joint space:

$$\theta_i(t), t \in [0, T] \tag{4}$$

Once we know the relative position of the object, expressed in 3D Cartesian coordinates $P_{object} = [xyz]$, we apply inverse interpolation to calculate what joint angles the arm must adopt to reach the desired position, such as the object's grip point. For this, inverse kinematics solves a nonlinear optimization problem or system of nonlinear equations that seeks to find the joint angles $\theta_1, \theta_2, ..., \theta_n$ such that the arm reaches the desired position:

$$f(\theta) = \mathbf{p}_{\text{efector}}(\theta) \stackrel{!}{=} \mathbf{p}_{\text{Objeto}} \tag{5}$$

Where

- f is the forward kinematics function of the system.
- $\theta = [\theta_1, \theta_2, ..., \theta_n]^T$ are the angles of the joints.

For its solution we resort to numerical or iterative methods, such as:

- Gradient descent method, Newton-Raphson method or Jacobian inverse / Pseudo-inverse
- Resolution through reinforcement learning or neural networks

In our case, we will use reinforcement learning, that is, we will not explicitly solve the inverse kinematics (IK) equations. Instead, our agent will learn how to move the arm joints to reach a goal, through trial and error with feedback. To do this, the system will make small changes to the joint angles, such as increasing or decreasing θ_i, and will receive a reward based on the negative distance to the goal:

$$r_t = - \left\| \mathbf{P}_{\text{efector}} - \mathbf{P}_{\text{objetivo}} \right\| \tag{6}$$

Enhanced Control Mechanisms

Given the structured state information provided by the simulation environment, we designed a control system that enhances the robot's ability to interact smoothly and accurately with its surroundings. Instead of relying on visual sensors, the robot uses precise numerical data—such as joint angles, end-effector position, and object coordinates—directly obtained from the simulator.

To ensure smooth grasping and transport, we implemented a dynamic speed adjustment mechanism. The robot modulates the velocity of its end-effector based on the distance to the target object and to the goal position. As the gripper approaches the object, the movement slows down to increase precision and reduce the risk of overshooting or knocking over the target.

During object manipulation, simulated force feedback is incorporated to ensure safe gripping. Although no physical pressure sensors are present, we estimate contact conditions by monitoring the relative positions of the gripper and object, along with the success or failure of grasp attempts. If the gripper closes but the object remains static, the system interprets this as a failed grasp and penalizes the agent accordingly in the reward function.

Furthermore, we introduce a trajectory planning module that operates in Cartesian space. It generates smooth intermediate waypoints for the arm to follow, ensuring stable and predictable motion between grasping and release phases. The joint acceleration and velocity constraints are also taken into account to avoid abrupt movements that could destabilize the object.

These enhanced control mechanisms, although built entirely on simulated state data, significantly improve the robot's ability to handle diverse manipulation tasks. They enable the agent to perform smooth, adaptive, and goal-oriented motions without the need for external cameras or additional sensory hardware.

Stability and Convergence

Some criteria we define for agent stability are:

- Its behavior must be well-constrained; the arm should not attempt to grasp objects outside its range of action, apply excessive force to the grip, or move at excessive speed. These parameters, when constrained, incorporate the BIBO stability of cybernetic control.

The success rates defined for each phase should show sustained improvement. Furthermore, the policy and value function must stabilize over time as the arm performs similar tasks (grasping, moving, releasing) in a repeatable and consistent manner. So we expect the robotic arm to behave as shown in Figure 8.

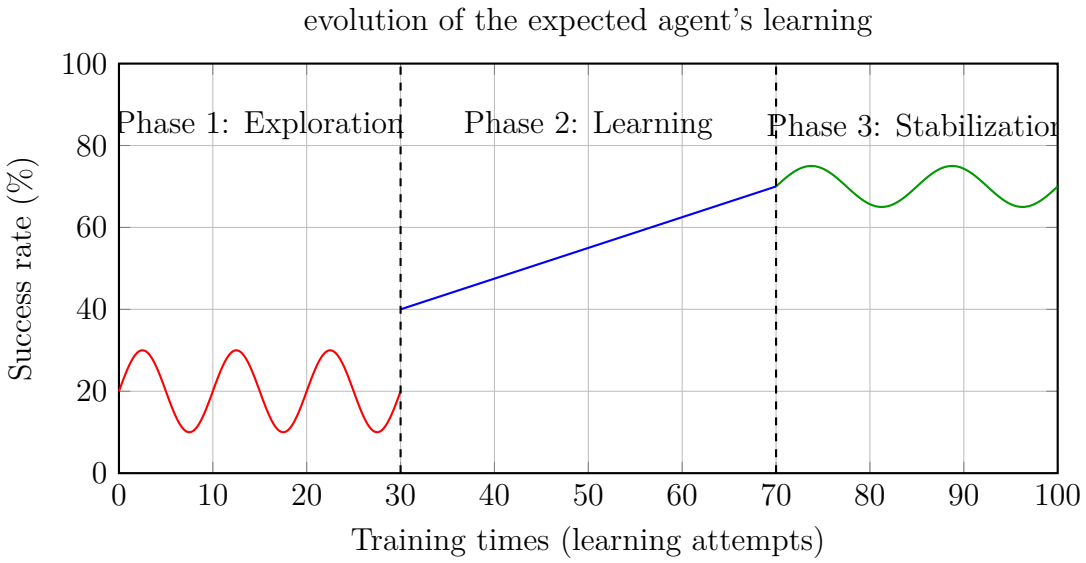


Figure 9

Evolution of the agent's success rate throughout training

Iterative Design Outline

Project Plan.

Phase	**Aim**	**Main Activities**	**Tools/Techniques**	**Success Criteria**
1. Conceptual Design and Require-ments	Define the technical and operational bases of the system	- Establish functional and non-functional requirements - Identify operational limitations	Requirements analysis Preliminary modeling	Complete and validated requirements
2. Perception and Sensing	Allow the system to perceive the environ-ment	- Normalization of state variables - Identification of graspable objects and positions	OpenCV, TensorFlow, Gymnasium Robotic	Detection accuracy $> 80\%$ Reliable classification
3. Kinematic Control and Motion	Execute precise and safe movements	- Modeling with homogeneous and D-H matrices - Inverse kinematics - Path planning	Python, NumPy, ROS, algoritmos RRT*	Collision-free trajectories Accuracy within the margin
4. Rein-forcement Learning	Optimize decisions with feedback	- Modeling the environment - Rewards per phase - SAC training	Gymnasium, Stable-Baselines3, SAC	Convergent policy Continuous performance improvement

Phase	Aim	Main Activities	Tools/Techniques	Success Criteria
5. Integration and Testing	Validate the system in a controlled environment	- Simulator tests	Gazebo, ROS,	Complete tests Success metrics show operating fluidity
6. Fault Tolerance	Ensure robustness against failures	- Fault detection - Recovery policies - State supervisions	Watchdogs, fallback sensores, ROS topics	Safe Reset Continuous operation
7. Evaluation and Optimization	Continuously measure and improve the system	- Evaluate metrics by phase - Retraining if necessary	Learning curves, entropy analysis	Success rate > 80% System stability

Table 1

Implementation strategy

Machine Learning Implementation

Algorithms and Frameworks

As a reinforcement learning algorithmwe chose to implement the Soft Actor-Critic (SAC) algorithm due to its approach based on nonlinear optimization models and stochastic policies, making it especially suitable for dynamic and uncertain environments such as those encountered in robotic arm control. This algorithm is based on the formulation of the smoothed Bellman equation, integrating an entropy term that

maximizes both the expected reward and exploration, which is essential for the agent's continuous adaptation to variations.

$$J(\pi_{\theta)} = E_{\pi_\theta}[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t) + \alpha H(\pi(.|s_t))] \tag{7}$$

Donde:

- $J(\pi_{\theta)}$ It is the policy that is being learned
- E_{π_θ} Mathematical expectation on the trajectories generated by the policy π_θ
- γ^t Discount factor that reduces the importance of future rewards
- $R(s_t, a_t)$ Reward received upon performing the action a_t in the state s_t.
- $H(\pi(.|s_t))$ represents the entropy of the policy π in the state s_t.
- α Coefficient that controls the importance of entropy in relation to reward.

This approach is particularly useful for addressing uncertainty factors and success rates identified in the system's phase diagrams, allowing the agent to make decisions that balance the exploitation of successful policies with feedback from new observations, achieving progressive improvement in the execution of complex tasks such as object identification, grasping, transport, and release.

For the development of the autonomous and adaptive robotic arm, we will rely on structured state-based input data provided by the simulation environment. This includes variables such as the position and orientation of the objects, the current joint angles of the arm, the state of the gripper, and other dynamic parameters. Since our control problem involves continuous action spaces and requires stability and sample efficiency, we have selected the Soft Actor-Critic (SAC) algorithm as the core of our learning agent.

To support this architecture, we will use Stable-Baselines3, a reliable and modular reinforcement learning library that provides an efficient implementation of SAC, including policy updates, replay buffer management, and entropy tuning mechanisms.

The agent's policy and value networks will be implemented using PyTorch, which provides the flexibility to define and adjust neural architectures that can process the structured observation vectors from the simulation. This combination allows for scalable, end-to-end training of the robotic arm in both simulated and real scenarios, without relying on image-based input.

Gym will serve as the simulation interface, allowing us to define the environment where the robotic arm interacts with objects.

Therefore, by selecting Stable-Baselines3 and PyTorch, we are fulfilling the requirement to implement modern deep reinforcement learning algorithms using scalable frameworks that integrate well with Gym compatible environments and structured numerical observations. This setup enables the robotic arm to learn optimal control policies directly from low-dimensional state variables like positions, velocities, and object coordinates without relying on visual input. The SAC algorithm, in particular, is well-suited for continuous control tasks, providing efficient and stable training for robotic manipulation.

On the other hand, we will implement the tool Poetry to manage our dependencies. Poetry will help us install and maintain the required libraries while avoiding version incompatibilities, even when working with multiple frameworks and packages.

Agent Testing and Evaluation

Experimental Setup

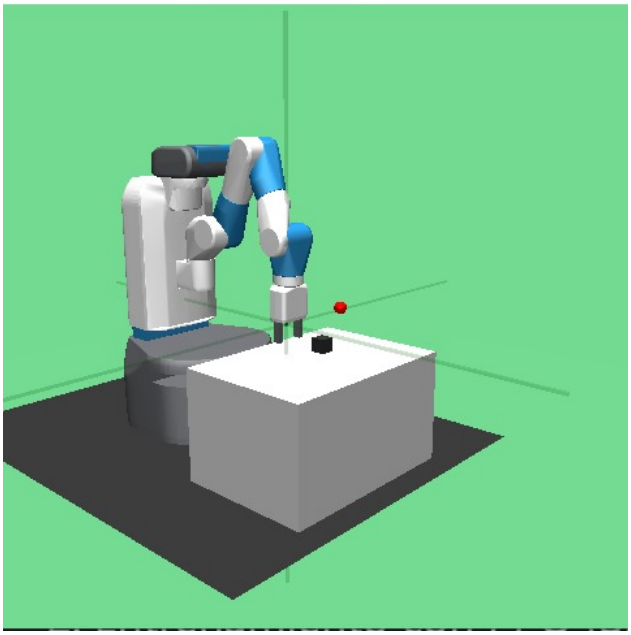
To train the robotic arm effectively, we designed a series of simulated scenarios that gradually increase in complexity. These scenarios are executed in a controlled environment using Gym-compatible simulations, where the agent interacts through a structured state space instead of image-based input.

Initially, the environment will provide simplified and noise-free observations that include key state variables such as:

- Joint angles and velocities of the arm
- Position and orientation of the target object (in Cartesian coordinates)
- Distance between the gripper and the object
- Gripper status (open/closed)

The best ally to make the simulations is gymnasium robotic, due to it make an perfect scenario about the robot arm and start to training according the accuracy we want for the testing

So, for the setup of the enviroment we are using, we need some installations such as, pyenv, gymnasium robotics, jupyter-lab and stable-baselines3. The libreary gymnasium robotics is installed directly from github, due to we had some problems from cmd. The objects the arm grap are not above the table, so, the training will be failed because of there are no rewards



From this simulation from the enviroment that give us we could start with the training with SAC (Soft Actor Critic), this algorithm consist in two parts, the first one who

is the actor, who make the learning and a critic which start to give the feedback and reward to the actor

The first training session was conducted using the Soft Actor-Critic (SAC) algorithm. SAC is a widely used reinforcement learning technique known for its stability and sample efficiency in continuous control tasks. During this training, we reached a success rate of 0.03 (3%) after completing 160,000 steps. The total training time was approximately 78 minutes, with an additional 43 seconds likely consumed by model initialization and data processing overhead. Despite SAC's reputation for efficiency, the training felt relatively slow and resource-intensive, especially when considering the minimal performance improvement. The low success rate indicates that SAC alone may struggle in sparse-reward environments, where successful outcomes are infrequent and harder to learn from.

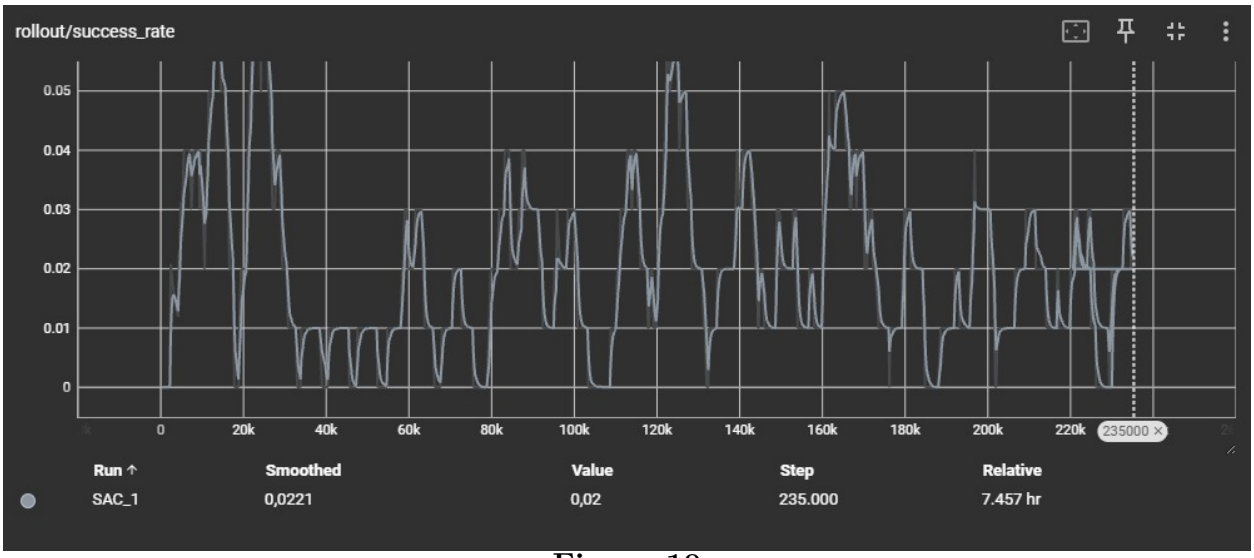


Figure 10

Graph SAC

Following this, we implemented a more advanced approach by combining SAC with Hindsight Experience Replay (HER). HER is particularly useful in reinforcement learning environments where agents receive sparse or binary rewards. It works by modifying unsuccessful trajectories during training, allowing the agent to learn from what would have

been successful under different goals. This technique is especially beneficial in goal-conditioned tasks, such as robotic manipulation. After training this combined SAC + HER model for 30,000 steps over a period of nine minutes, we observed a slight improvement with a success rate of 0.04 (4%). While the increase was marginal, it suggests that HER may be helping the agent extract more meaningful learning signals from the environment, even with fewer steps. However, the overall performance still indicates that the environment remains a challenging one, and further optimization or longer training may be needed to see significant gains.

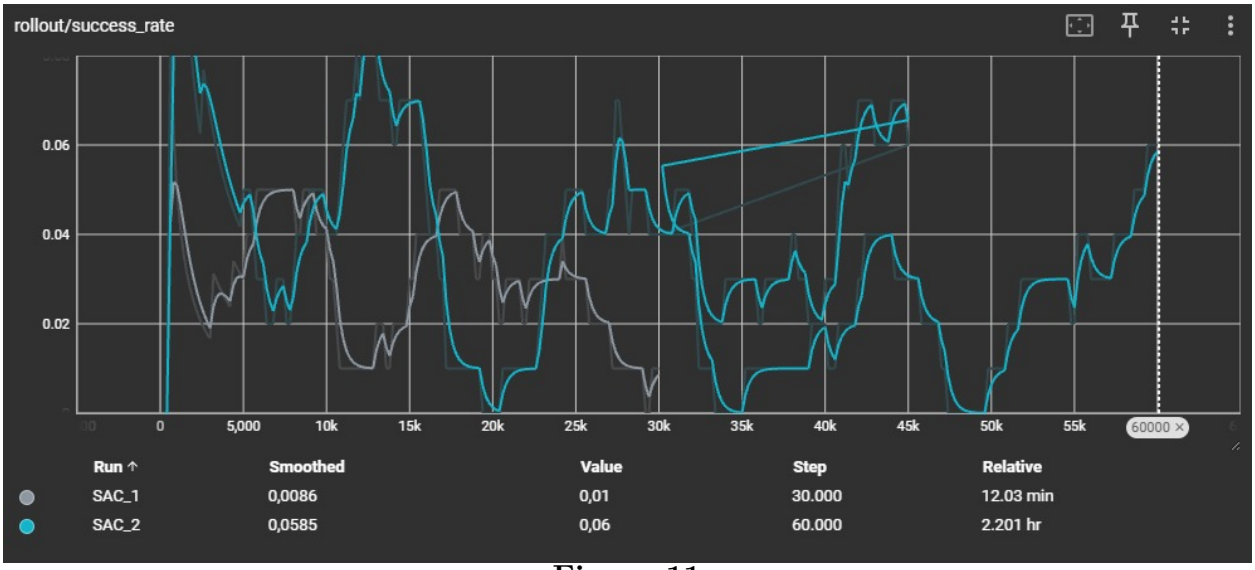


Figure 11

Graph SAC+HER

Next, we explored the integration of Deep Deterministic Policy Gradient (DDPG) with HER. DDPG is an actor-critic algorithm specifically designed for continuous action spaces, where decisions require fine-grained control rather than discrete choices. This makes it particularly suitable for tasks like robotic arm manipulation, where smooth, precise movement is critical. The actor network is responsible for selecting actions, while the critic evaluates them, allowing for more informed and stable learning. Combining DDPG with HER aims to leverage both the strength of DDPG in continuous control and the benefit of

HER in sparse-reward settings. This pairing is often used in real-world robotic applications, where learning from rare successful experiences is crucial for effective training.

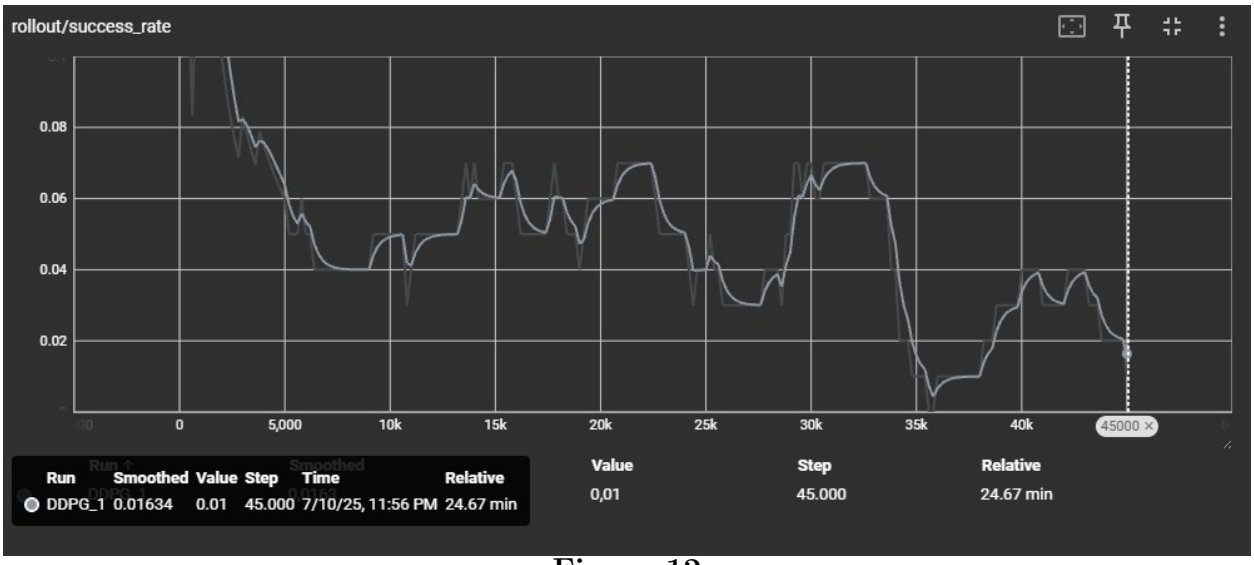


Figure 12

Graph DDPG

Performance Metrics

To make sure our robotic arm is actually learning and improving during training, we are going to collect and compare several key metrics and the best way to do that is by plotting graphs that show how the agent behaves over time.

The most important graph we will use is the learning curve, which shows the total episode reward (the total reward the agent gets in each episode). At the start of training, this curve usually looks pretty noisy — rewards go up and down like waves, as seen in Figure 12.

That happens because the agent is exploring the environment, trying random

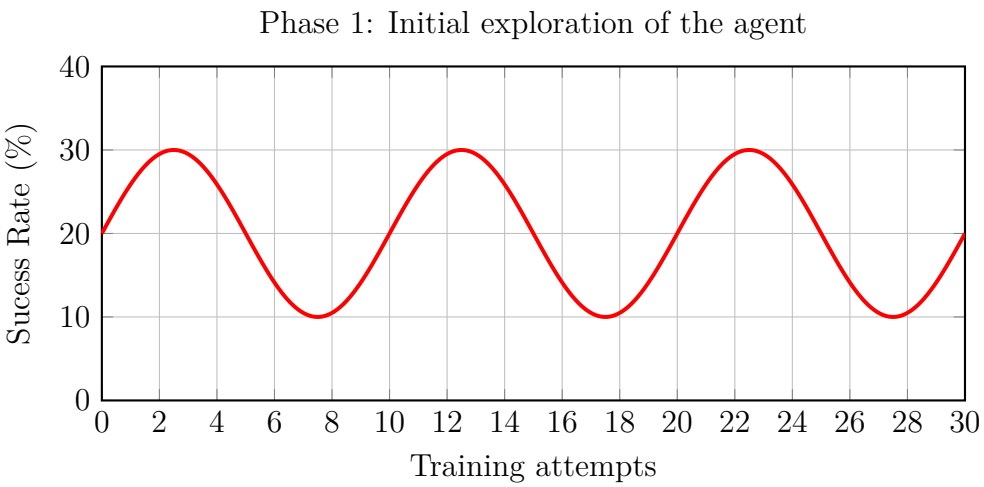


Figure 13

learning during the first phase

actions, and has not learned what works yet.

Over time, if everything is working well, we should see the rewards start to go up on average, as seen in Figure 13, even if there are still some drops along the way.. This is a good sign that the robotic arm is learning to detect objects better, move more efficiently, and complete tasks without errors.

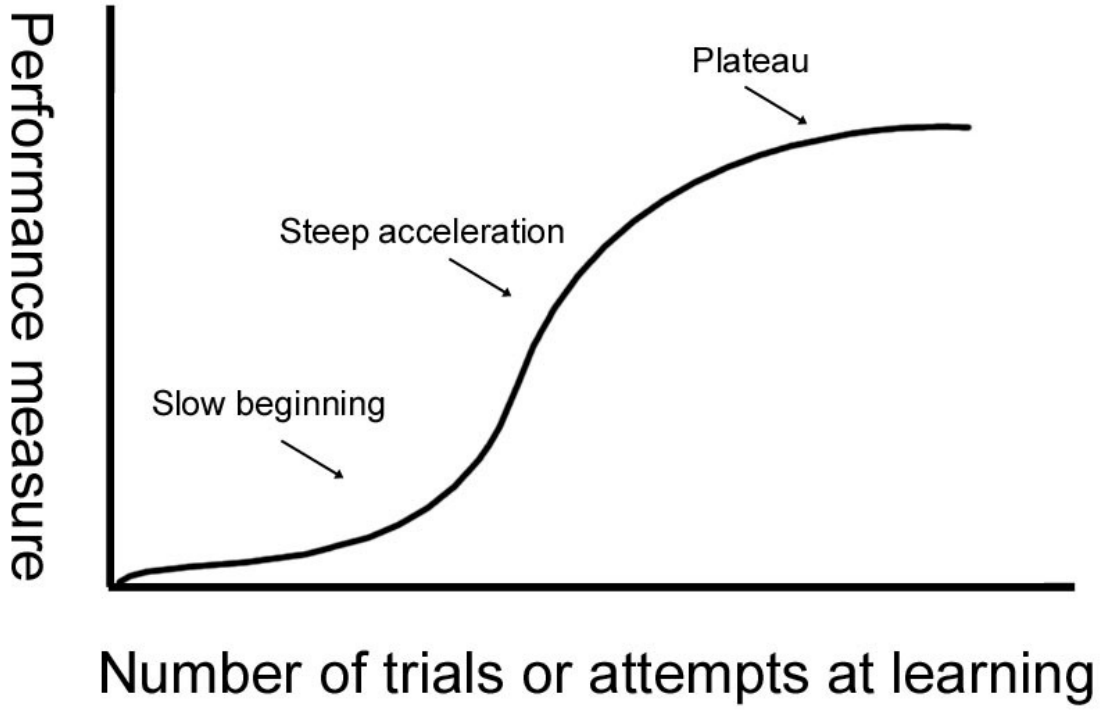


Figure 14

Curve Learning Phase:2

We will also pay attention to how fast the reward curve stabilizes, which tells us the convergence speed. A fast convergence means the agent is learning quickly and consistently. If the curve flattens out at a high reward level, that usually means the policy is solid and the robot is doing its job well. On the other hand, if the curve stays flat at a low level, or keeps jumping around without improving, we will know that something's wrong maybe with the reward function, the neural network, or the input data. Here an example about a flat curve:

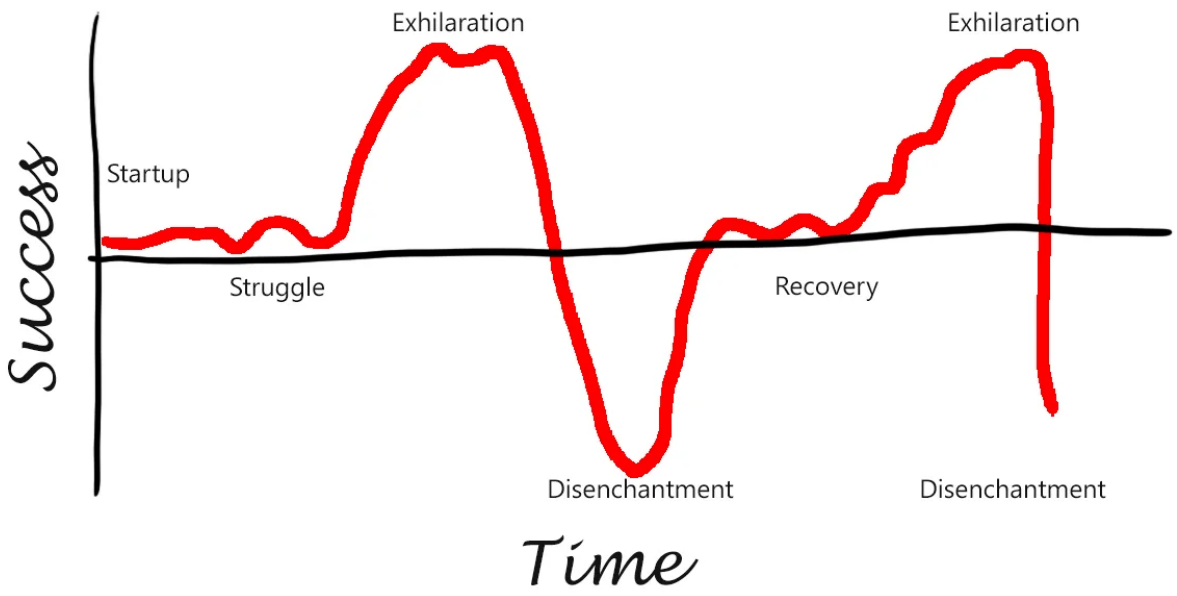


Figure 15

Flat Curve

So we finally hope to observe behavior like that seen in Figure 10, in the development of our project, seeking to make the last phase as stable as possible.

Another helpful graph is the moving average reward, which smooths out the short-term ups and downs and makes it easier to see the overall trend. This helps us avoid getting distracted by random noise in the data.

Then, we might track other metrics, like loss values, Q-values. All these can give us clues about what the model is learning and whether it is overfitting, underfitting, or behaving in unexpected ways.

These visualizations are too useful because they help us understand the agent's progress, we will make the graphics with matplotlib.

Mono-agent

The autonomous robotic arm is considered as a single-agent due to its function depend on just one agent, who take decisions and interact with the enviroment, the robotic arm act, learn and receive reward, it doesn't wait the answer from another agent, all its experience is from its previous steps, Although, there is a way to get multi-agent in this project, making agents from each articulation from the arm, but is better with just one agent due to all joints are controlled by a single model or decision policy, the agent (the RL model) decides on a joint action, the reward is shared and calculated based on the final result, not individually for each axis.

Timeline Project

Week	**Tasks**
Week 0	W1 Analyze Uses of Cases and Components
Week 1	Mathematical Model
Week 2	Refinement Feedback - Loop Agent Stability
Week 3	Framework implemented
Week 4	Simulation Parameters - Scenario variations
Week 5	ML integration
Week 6	Cybernetic Control Mechanisms
Week 7	Finalize and Refine Feedback
Week 8	Self-regulation, adaptability and resilience
Week 9	Test Cases/Evaluation
Week 10	Environment Setup
Week 11	Agent Definition - RL implementation
Week 12	Visualization - Metrics
Week 13	Testing & Validation - Record Video

Referencies

Amin, S. (2024). Deep q-learning. *Medium*. Retrieved from https://medium.com/@samina.amin/deep-q-learning-dqn-71c109586bae

Chizari, M. (2025). Which framework do i choose? *Linkedin*. Retrieved from https://www-linkedin-com.translate.goog/pulse/which-framework-do-i-choose-tensorflow-keras-pytorch-mohamed-chizari-zbdwe?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=rq#:~:text=Why%20Choose%20Keras%3F,scalability%20while%20maintaining%20its%20simplicity.

Chizari (2025) Amin (2024)