Simulation of an autonomous and adaptive robotic arm

Juan Diego Lozada

Cristian Santiago López Cadena

Workshop 3

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

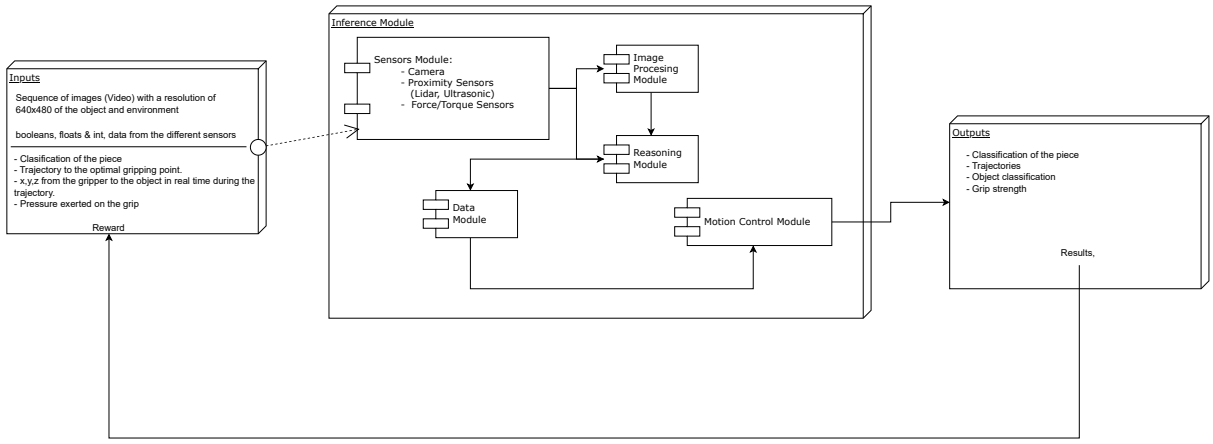

Figure 1

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

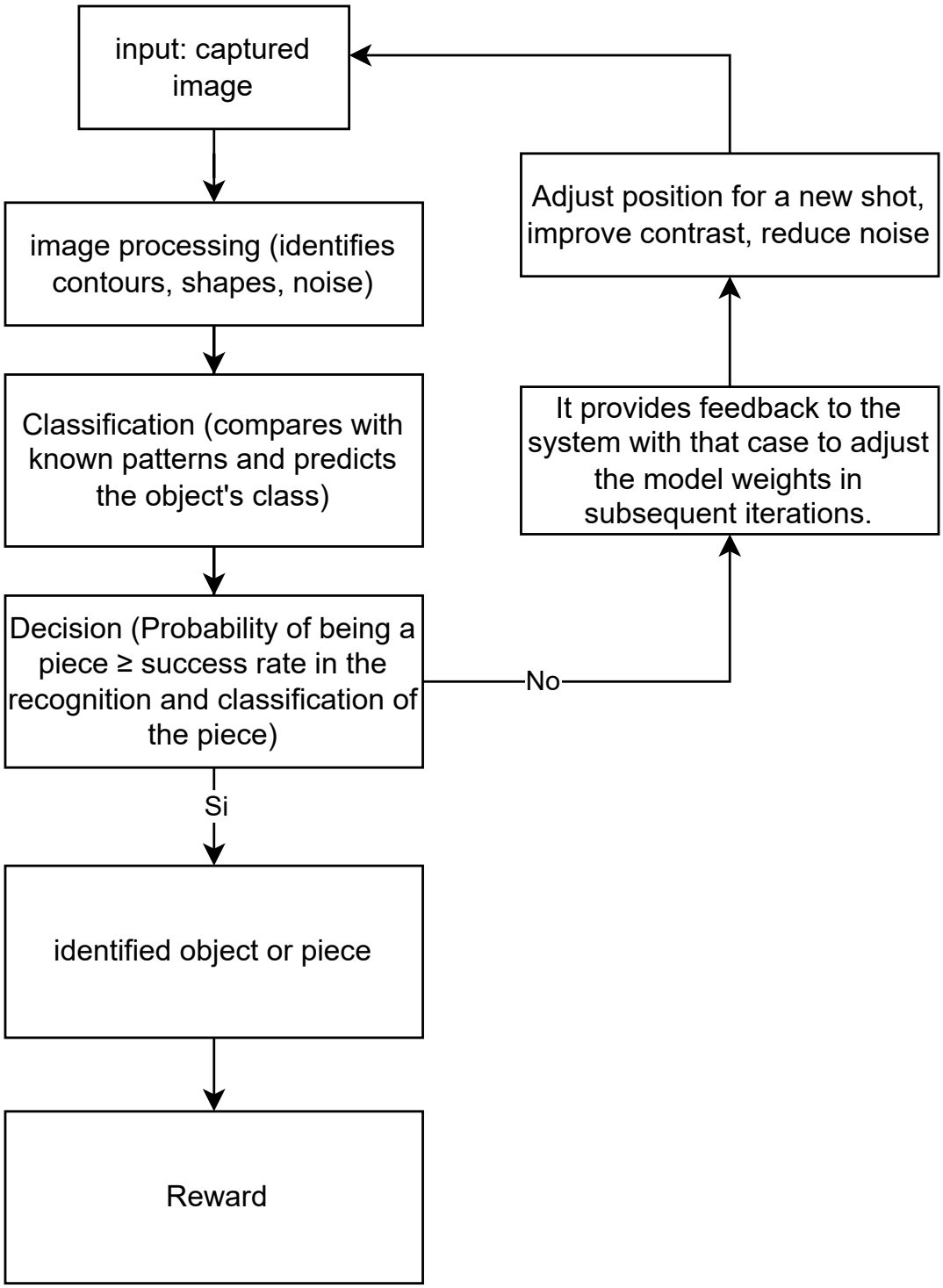

Figure 2

feedback loop identify object/part

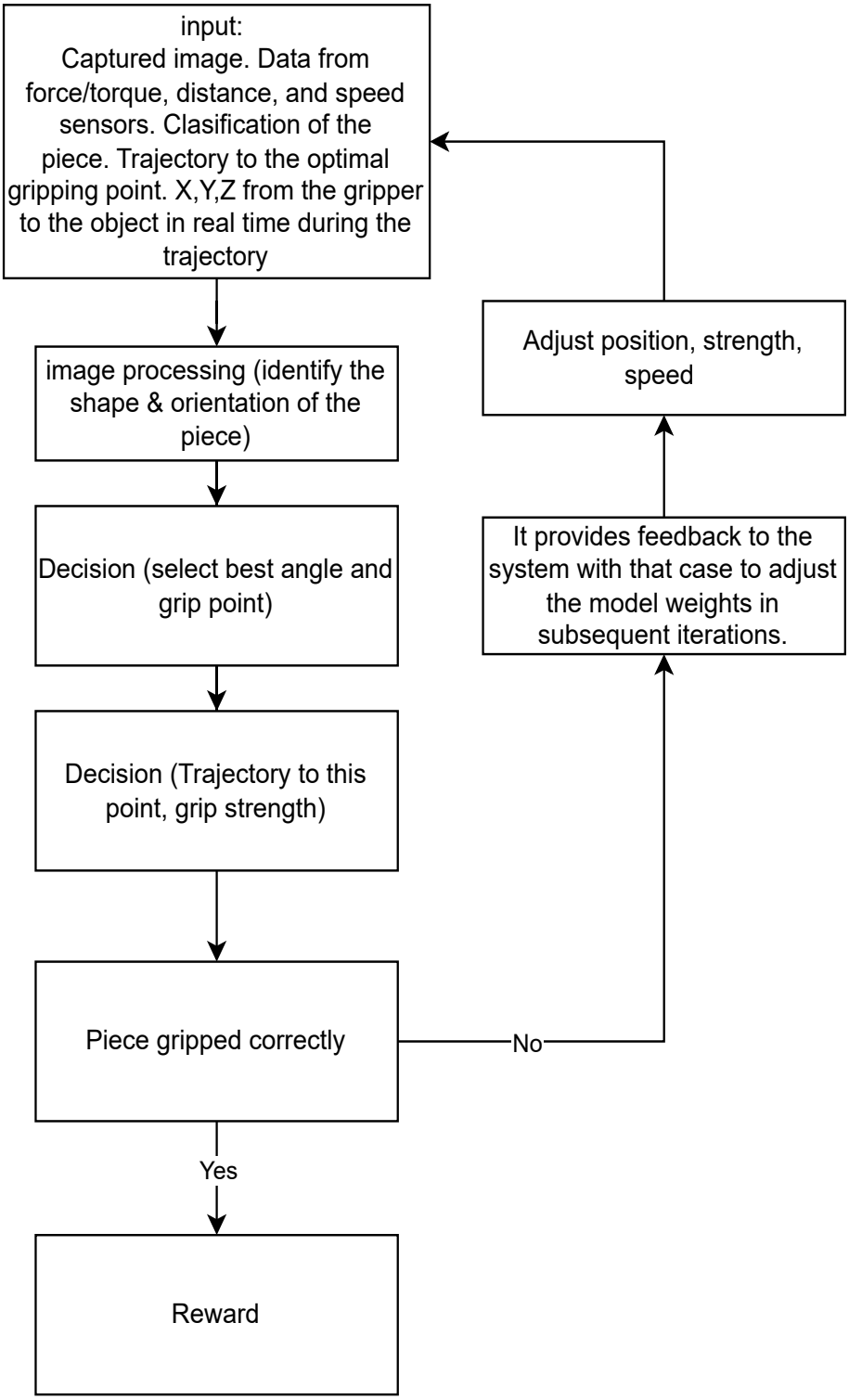

Figure 3

feedback loop grab piece/object

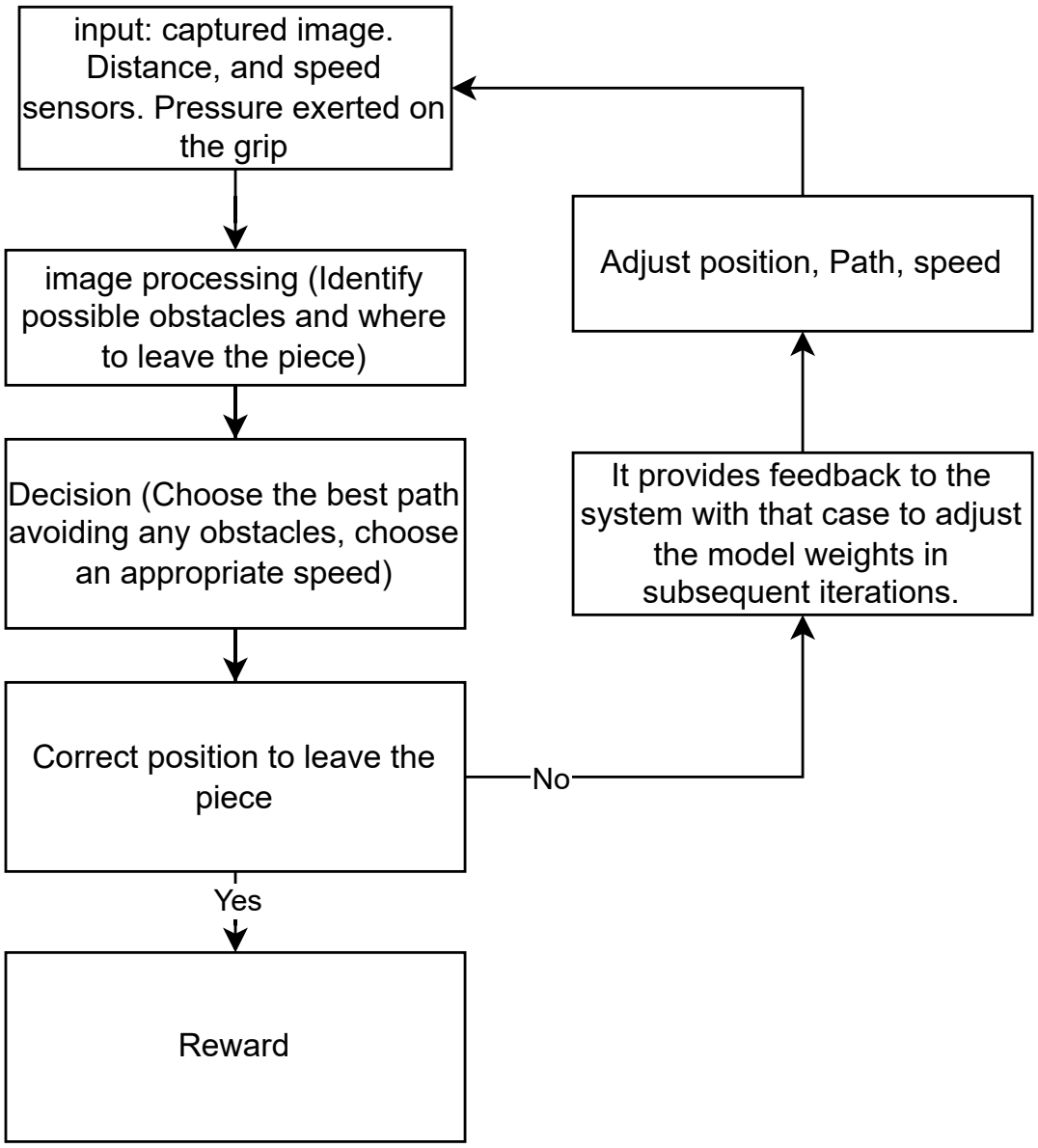

Figure 4

feedback loop transporting part/object

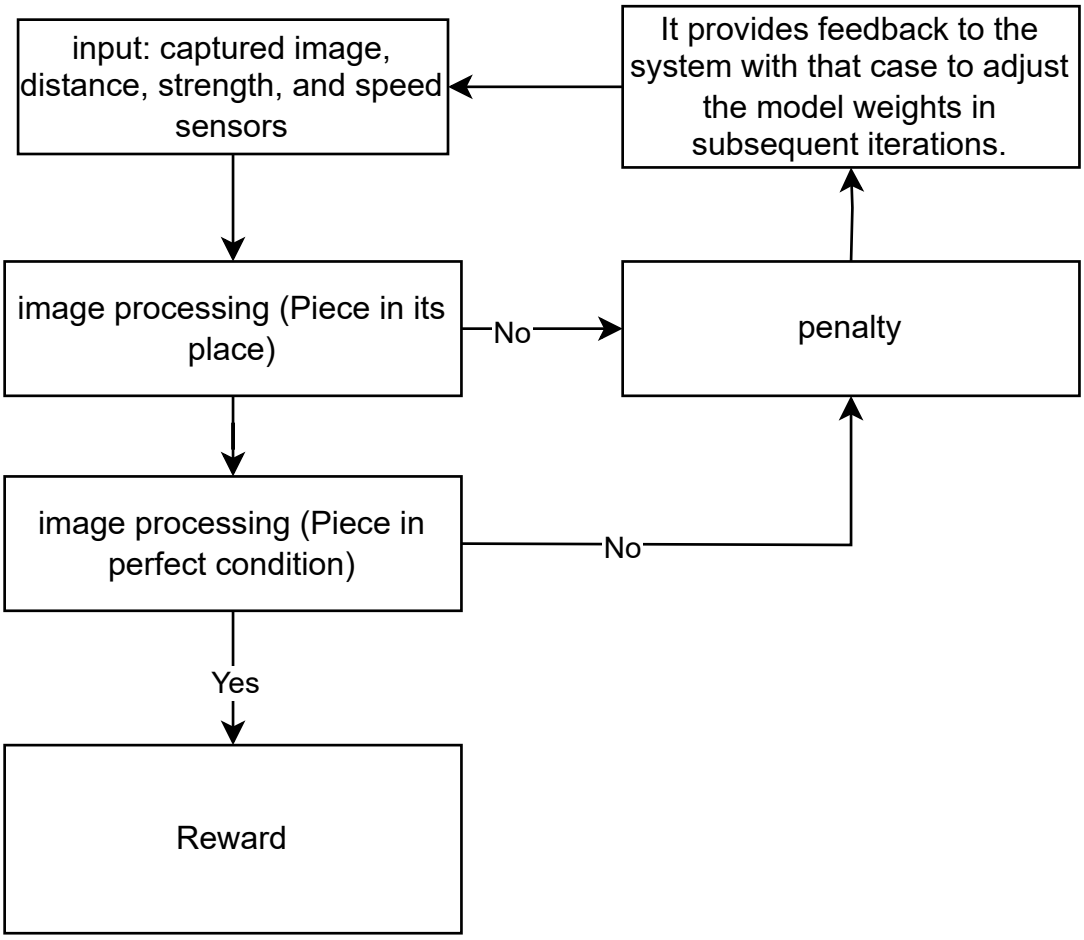

Figure 5

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. Among these options, we find models that use image recognition algorithms such as SSD – Single Shot MultiBox Detector, YOLO – You Only Look Once, Faster R-CNN, and Mask R-CN. Each of these has its advantages and disadvantages, so we are still studying which one is most suitable for our project.

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 begins by visually processing the work environment. Using computer vision techniques, the parts present are identified, their relative locations are determined, and morphological classification (size, shape, material, etc.) is performed.

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

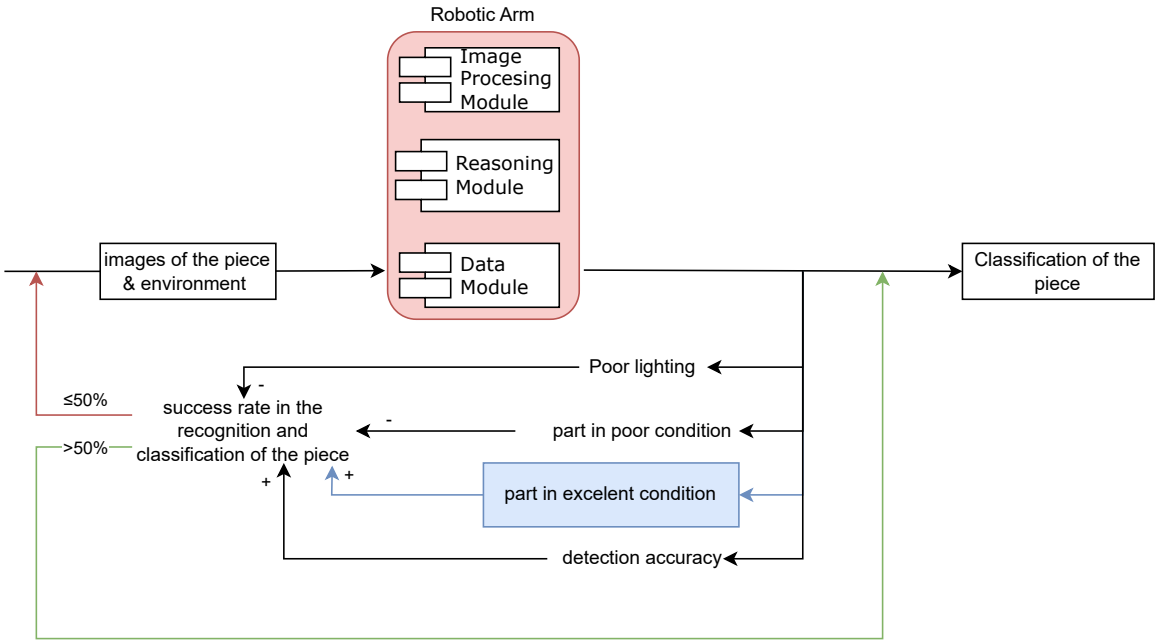

Figure 6

Environment and object recognition 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 each phase of the system's operation. These factors include:

- Poor lighting: This factor affects image quality, as it can generate shadows or noise that make it difficult for the system to recognize the object or its position.

- Part in poor condition: If the part is in poor condition or broken, its geometry and appearance change, which can confuse the system and prevent it from recognizing the part.
- Part in excellent condition: This factor positively affects the system's accuracy because if the part is in perfect condition, the system will have higher accuracy in part classification.
- Detection accuracy: If image recognition accuracy is poor, it can generate errors in part classification and recognition, which would negatively impact the system.

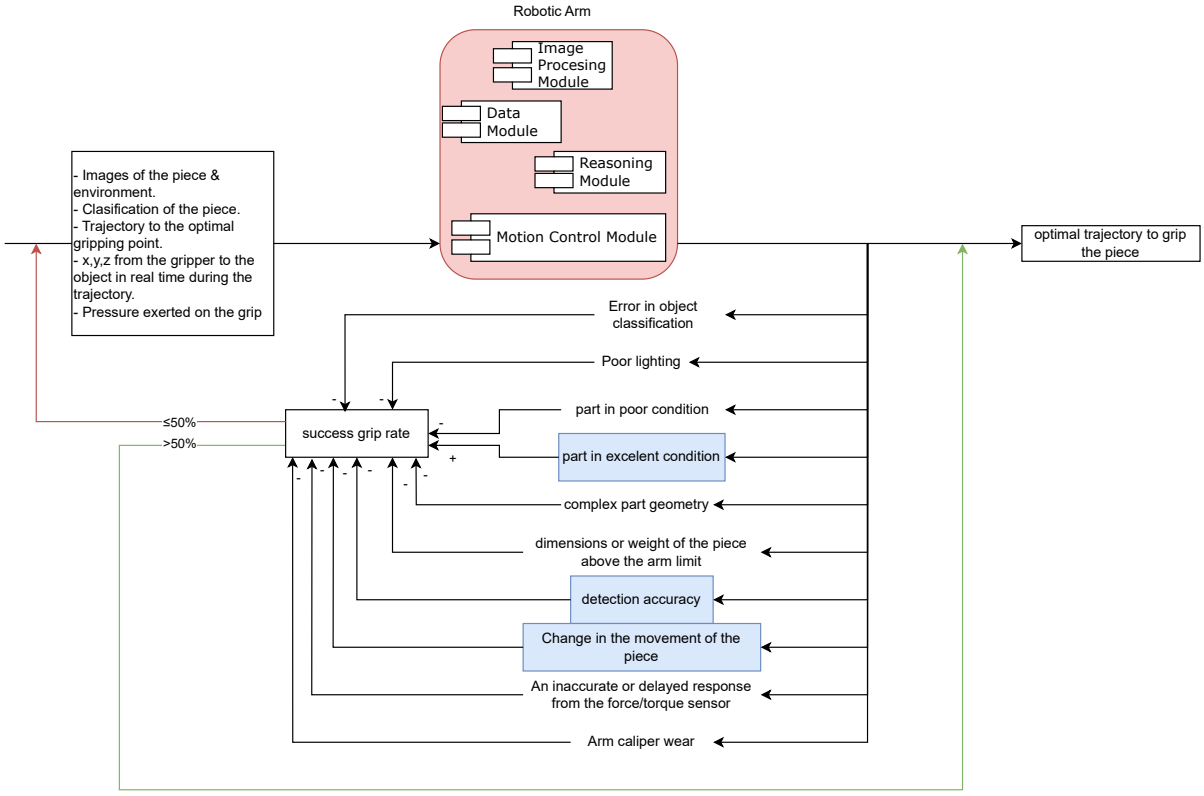

Figure 7

Grasp planning and execution phase

As evidenced in the phase diagram presented above (Figure 7), 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.

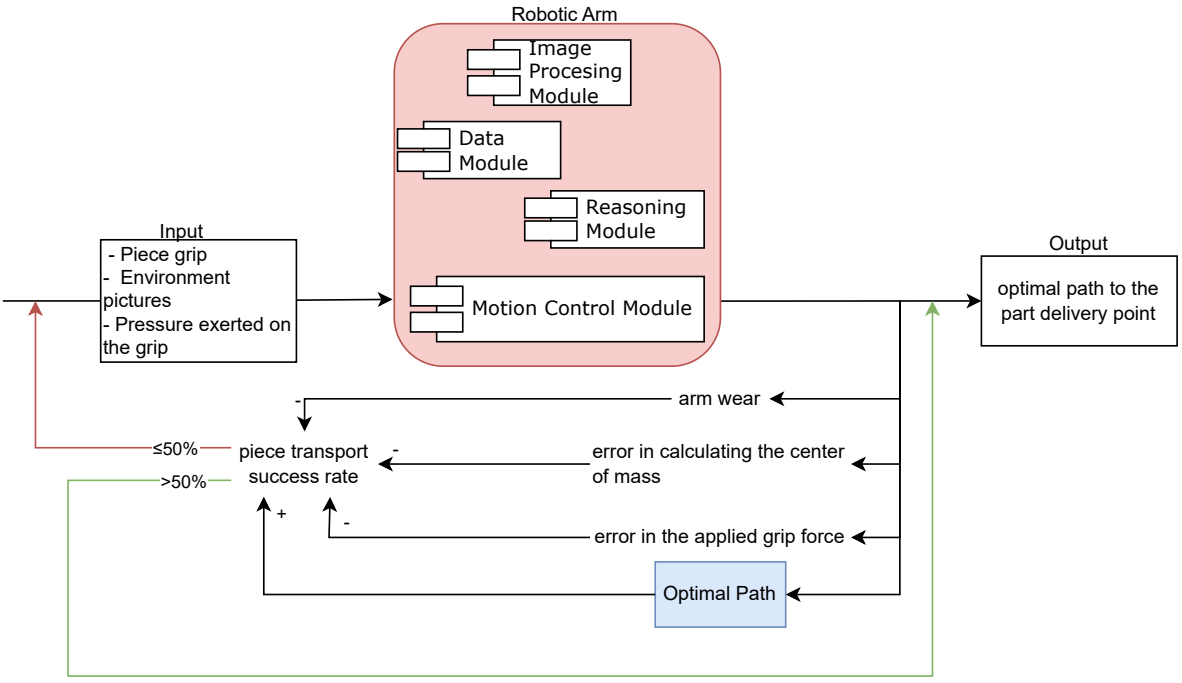

Figure 8

Part transport 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 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.

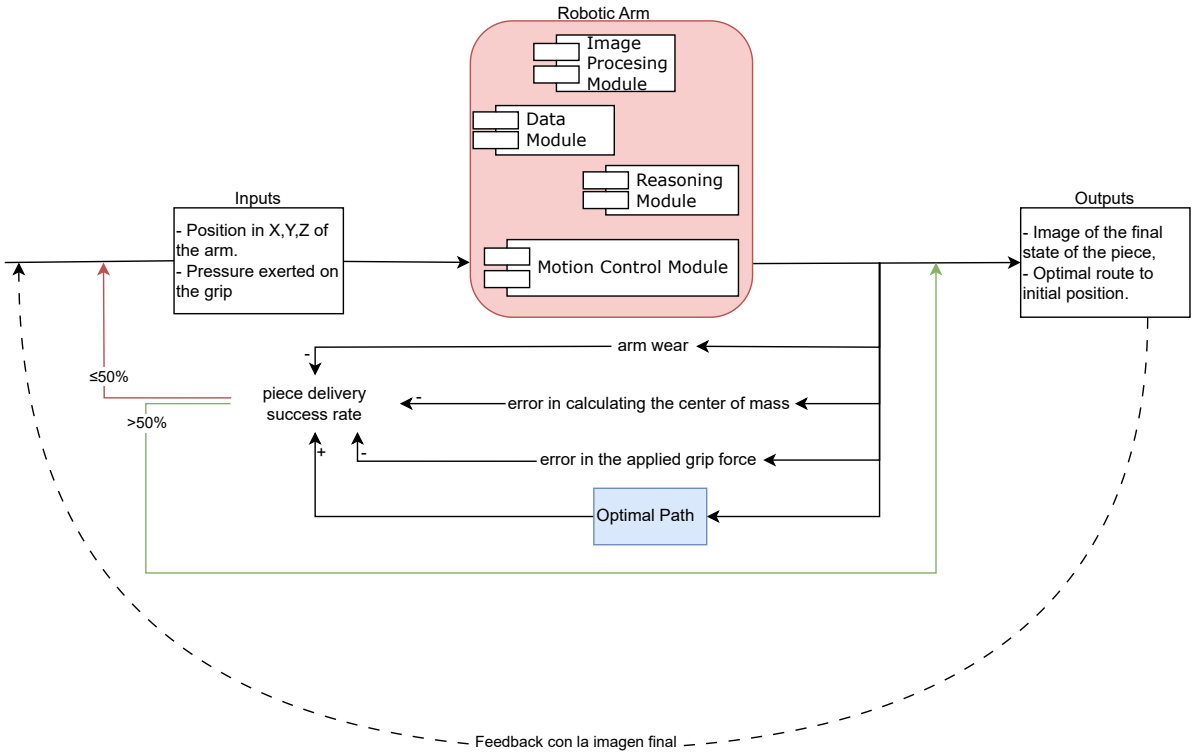

Figure 9

Release and verification phase

As evidenced in the phase diagram presented above (Figure 9), 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 recognition and classification success rate (Phase 1, Figura 1).
2. Piece grip success rate (Phase 2, Figura 2).
3. Piece transport success rate (Phase 3, Figure 3).
4. Piece delivery success rate (Phase 4, Figura 4).

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:

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

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.

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

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

After reviewing the sensors, we decided to install two RGB-D cameras one in the handle and one outside the arm. This sensor not only allows for an in-depth analysis of the object in front of the robot, but it also allows us to identify the object's trajectory in real time and validate the object's condition once it's delivered. We also decided to incorporate acceleration sensors in the arm structure to adapt the speed to the transport process.

On the other hand, we will use a projection matrix to convert 3D world coordinates into 2D image coordinates based on the intrinsic (describe the intern part from the camera and how it show the points of three-dimensional space to the image plane) and extrinsic (describe the position and the orientation from the camera in a 3D) parameters of the RGB-D camera. This enables accurate localization of objects in the camera frame, crucial for computing grasp points and motion planning.

The intrinsic Parameters are the next one:

- Focal Length (f)

- Principal Point (cx,cy)

- Skew, inclination factor

$$K = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{7}$$

Extrinsic Parameters

The transformation from a 3D point in the world (X, Y, Z) to a 2D point in the image (u, v) is:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

RGB-D frames will be collected during simulation and real-world trials, consisting of synchronized RGB and depth images. These will be annotated with class labels, center-of-mass coordinates, and segmented masks. Calibration data (camera intrinsics, homographies, and transformation matrices) will also be stored to support 3D reconstruction and labeling.

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 10.

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

Phase	**Aim**	**Main Activities**	**Tools/Techniques**	**Success Criteria**
2. Perception and Sensing	Allow the system to perceive the environment	- Sensor integration (RGB-D cameras, Force/Torque and Speed Sensors) - Classification with CNN - Centroid and Shape Calculation	OpenCV, TensorFlow, sensors RGB-D	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. Reinforcement Learning	Optimize decisions with feedback	- Modeling the environment - Rewards per phase - SAC training	Gymnasium, Stable-Baselines3, SAC	Convergent policy Continuous performance improvement
5. Integration and Testing	Validate the system in a controlled environment	- Simulator tests	Gazebo, ROS,	Complete tests Success metrics show operating fluidity

Phase	Aim	Main Activities	Tools/Techniques	Success Criteria
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.

$$J(\pi_{\theta)} = E_{\pi_\theta}[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t) + \alpha H(\pi(.|s_t))] \tag{8}$$

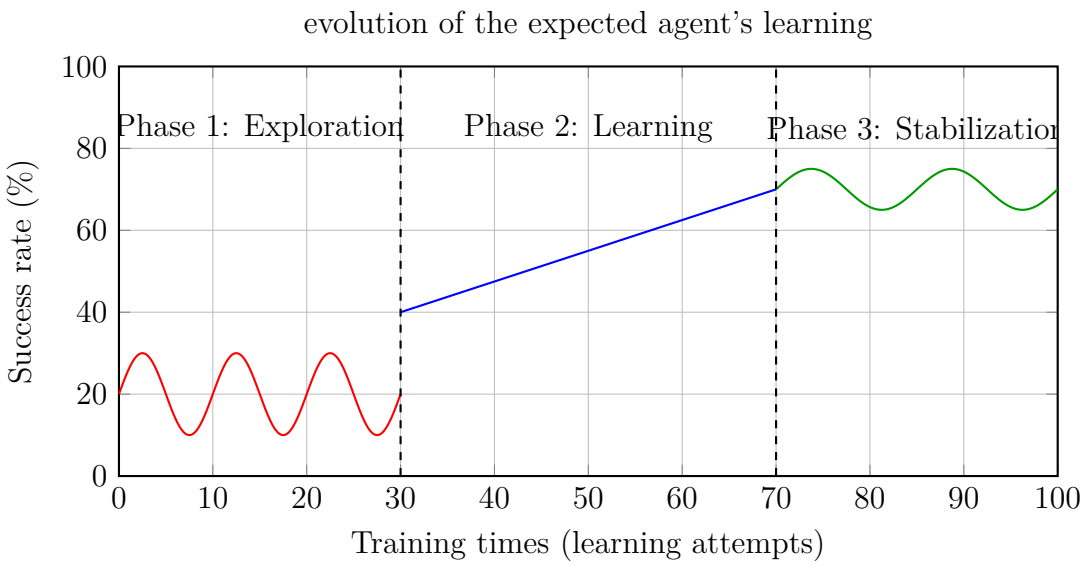

Figure 10

Evolution of the agent's success rate throughout training

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 image-based input data processed through convolutional neural networks (CNNs). Therefore, we need frameworks that support image input, deep learning architectures, integration with OpenAI Gym environments, and provide pre-implemented Deep Q-Networks (DQN) to speed up development. Additionally, scalability and flexibility are essential to adapt the system as it evolves.

The tools that best meet these requirements are Stable-Baselines3 and PyTorch.

Stable-Baselines3 will be used to implement the DQN algorithm, taking care of the reinforcement learning logic, agent training loop, and exploration strategies. It offers a modular, tested implementation that saves development time.

PyTorch will be used to define custom CNN-based policies that process visual input (camera frames), extract features, and estimate Q-values. Its flexibility will allow us to design architectures suited to the robotic arm's perception needs.

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 Deep Q-Networks using modern, scalable frameworks that integrate well with Gym environments, image inputs, and deep learning capabilities. This setup will allow the robotic arm to learn from visual data and classify objects as part of its autonomous operation.

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

The scenarios we want to implement are designed to help the agent learn in the best possible way. At the beginning, we will use clean and ideal images for training. These

images will show the objects with good lighting, centered in the frame, and in clear and stable positions. The idea is to give the robotic arm the best visual conditions so it can focus on learning the task correctly from the start.

If we start with real world images that have bad lighting, objects in the corners, or too much background noise, the robot could get confused, and the training might give unexpected or poor results. That's why we will first train it with perfect images

Figure 11

Good Photo Apple

Later, once the robot arm performs well with clean images, we will move on to using more realistic photos, where the objects may not be centered, background noise and bad lighting. This will help the robot learn to deal with more difficult situations, like in real

environments.The next photo the apple is not centered, for that reason, it is not a choice put it on the training

Figure 12

Bad Photo Apple

In conclusion, we will first use perfect images to teach the basics, and then we will challenge the robot with real conditions so it becomes more robust and adaptable.

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.

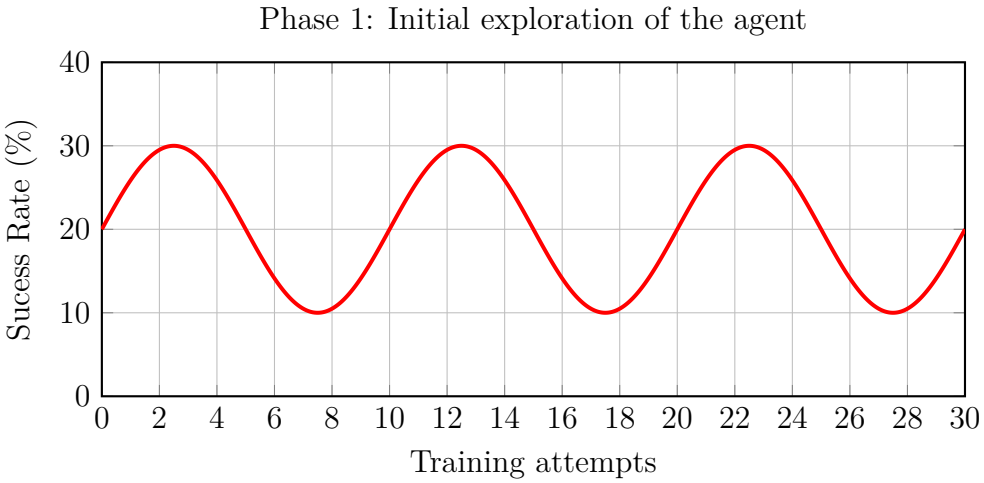

Figure 13

learning during the first phase

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

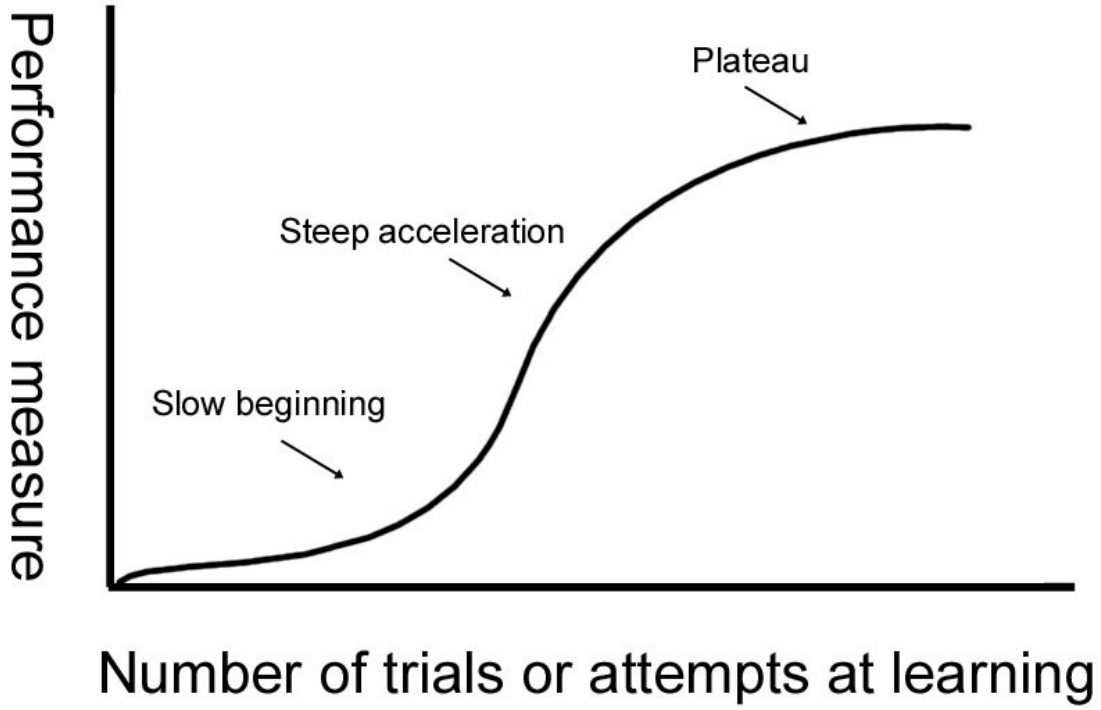

Figure 14

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:

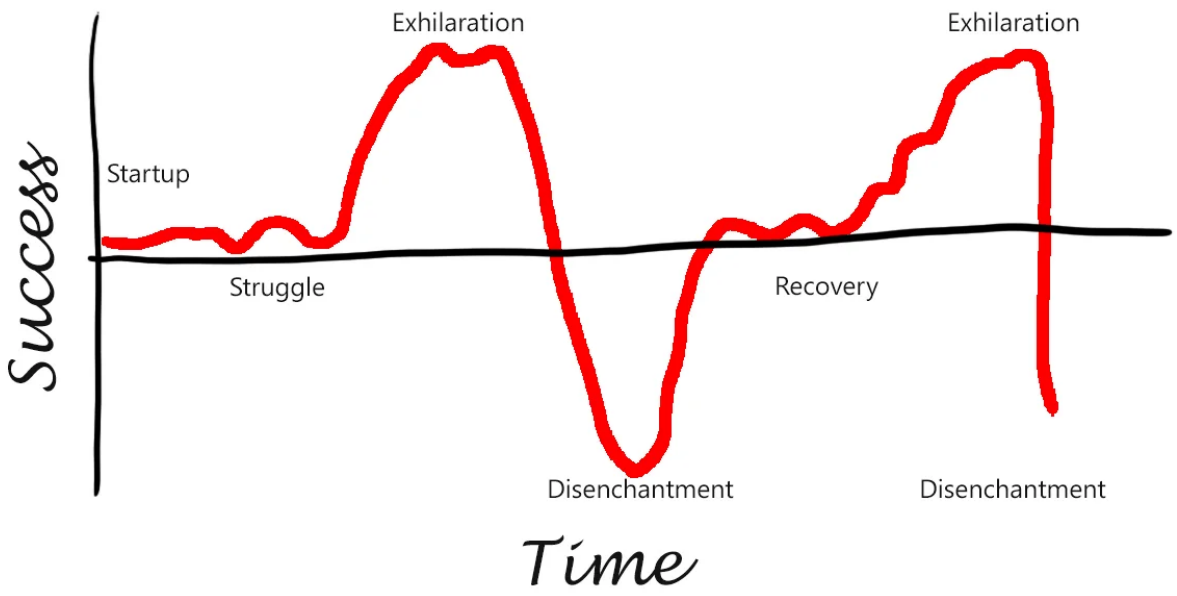

Figure 15

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)