MASTER EN INGENIERÍA EN TECNOLOGÍAS DE INDUSTRIALES / MASTER IN SMART INDUSTRY

FINAL MASTER'S PROJECT

Reinforcement learning applied to a Universal Robot with a vacuum gripper

Author: Álvaro Burgos Madrigal

Director: Philippe Juhel

Luxembourg / Toulouse

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título ..
...……..Reinforcement learning applied to a Universal Robot…….

……with a vacuum gripper……..……
……..…….……

……..…….……

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico ……. es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada

de otros documentos está debidamente referenciada.

Fdo.: Álvaro Burgos Madrigal Fecha: 17/08/2022

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Philippe JUHEL Fecha: 18/ 08/ 2022

MASTER EN INGENIERÍA EN TECNOLOGÍAS DE INDUSTRIALES

TRABAJO FIN DE GRADO

Reinforcement learning applied to a Universal Robot with a vacuum gripper

Author: Álvaro Burgos Madrigal

Director: Philippe Juhel

Luxembourg / Toulouse

Greetings

Thanks to my Director, for his support and thanks to my flat mates and my girlfriend for still wanting to know from me after cancelling my plans with them for months.

Reinforcement learning applied to a Universal Robot with a vacuum gripper

Author: Burgos Madrigal, Álvaro.

Director: Juhel, Phillipe.

Entidad Colaboradora: ICAM Toulouse

Palabras Clave: *Ventosa, Resnet, NN, CNN, Replay Memory, aprendizaje por refuerzo*

1. Introducción

Es incuestionable que la inteligencia artificial es uno de los retos clave de la sociedad de esta época. La inteligencia artificial ha demostrado ser adecuada para una gran variedad de casos de uso, desde ayudar en las decisiones orientadas al Big Data (por ejemplo, el marketing dirigido en las redes sociales), hasta proporcionar a un robot la capacidad de caminar, hablar o incluso jugar, llegando incluso batir a las propias personas.

Esta inteligencia, entre el resto de tecnologías que conforman la industria inteligente, está diseñada con dos objetivos principales. El primer objetivo es mejorar la eficiencia y eficacia de los procesos industriales actuales. Cuando un robot aprenda a hacer una tarea por sí mismo, a menudo será capaz de ofrecer el mismo resultado que un humano, pero en menos tiempo, con menos defectos y consumiendo menos recursos. Estas mejoras impulsarán la rentabilidad a largo plazo en las empresas. El segundo objetivo importante de la industria inteligente, y especialmente en el aprendizaje de refuerzo, es situar a la humanidad donde está el verdadero valor de la empresa. Un robot servirá sobre todo para eliminar las tareas repetitivas que un trabajador se ve obligado a hacer y que no contribuyen a dar sentido a su vida, y para ayudar al humano proporcionándole una mayor visibilidad allí donde las habilidades humanas no dan la talla.

El propósito de este proyecto sigue estos objetivos. El proyecto y el informe que se presentarán representan los siguientes pasos de un equipo de Investigación y Desarrollo de la universidad del ICAM, en Toulouse. El equipo comenzó a desarrollar este proyecto hace dos años, y su objetivo final es enseñar a un robot a recoger objetos utilizando la visión. El escenario que debe resolver el robot está formado por dos cajas, una cámara y el propio robot. Una de las cajas está llena de los mismos objetos y la otra está vacía. Entonces, el objetivo es que el robot detecte el punto con mayor probabilidad de ser apto para la recogida. Entonces, el robot colocará el vacío en el punto seleccionado y recogerá con éxito el objeto y lo colocará en la otra caja.

La principal complejidad del problema se debe a la falta de información sobre el objeto que se va a recoger. El algoritmo sólo tendrá acceso a una cámara de color, una cámara de profundidad y conocerá la profundidad del suelo.

Este proyecto, a la larga, podría tener varias aplicaciones. Entre ellas, este robot, entrenado correctamente, sería capaz de colocar algunos objetos, como componentes de un producto, tornillos, herramientas, etc., del inventario de una fábrica a la placa de un AGV que podría, una vez que el robot haya terminado de acercar estos artículos al operario, ahorrarle el tiempo y la incomodidad de tener que levantarse para ir del espacio de trabajo al almacén y llevar todos los artículos necesarios consigo.

2. Planteamiento del Proyecto

La versión anterior de la solución utiliza un algoritmo de aprendizaje supervisado para entrenar al robot, basado en tres fases. La primera es la de adquisición de datos, en la que el robot trata de elegir aleatoriamente puntos dentro de la caja llena de objetos y guarda una instantánea recortada de la cámara alrededor del punto seleccionado con una etiqueta de éxito/fracaso dependiendo de si el objeto fue elegido o no. La segunda es la fase de entrenamiento en la que se utiliza una Red Neuronal Convolucional Residual (Resnet CNN) y una Red Neuronal (NN) para calcular para predecir la probabilidad de éxito/fracaso de coger una pieza en un punto teniendo como entrada la imagen recortada del punto seleccionado. El Resnet ya está entrenado y, aunque no está adaptado explícitamente al proyecto, sirve para proporcionar las principales características de una imagen. La tercera es la fase de prueba, que en última instancia será la fase de producción, en la que se prueba el algoritmo para calcular la precisión.

Sin embargo, el proyecto que se explica en este informe explorará una solución diferente, utilizando el aprendizaje por refuerzo. La CNN seguirá siendo una Resnet preentrenada, pero la NN seguirá un algoritmo de aprendizaje por refuerzo, que convertirá las tres fases mencionadas anteriormente en un algoritmo dinámico.

Por otro lado, uno de los puntos débiles de las investigaciones del equipo del laboratorio a lo largo de los años ha sido tratar los problemas del robot, no el algoritmo CNN/NN en sí. Así, el proyecto se construirá bajo un gemelo digital del laboratorio de Toulouse, utilizando el software de simulación Coppelia, para que eventualmente los ciclos de aprendizaje sean más rápidos y no se dependa del hardware para avanzar en el proyecto.

Para ello, los 6 meses del proyecto se dividirán en tres partes igualmente distribuidas en el tiempo. Los dos primeros meses del proyecto se centrarán en la adquisición de conocimientos sobre el aprendizaje por refuerzo y Coppelia. Los dos segundos meses del proyecto se reservarán para desarrollar el software y el algoritmo. Por último, en los dos últimos meses se analizarán los resultados, se realizará un análisis de sensibilidad y se redactará el informe.

3. Descripción del modelo

La siguiente figura muestra el escenario construido en el software Coppelia. El robot UR5 está en el centro del escenario, y tiene dos cajas a ambos lados del robot, una llena de objetos y la otra vacía. Cada caja tiene dos cámaras diferentes, una RGB y otra que mide la profundidad. Estas cámaras se pueden ver en la parte superior izquierda y derecha de la figura.

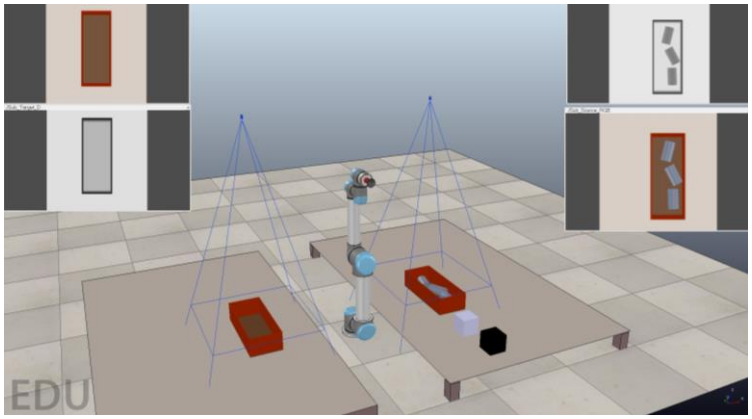

Figure 1 Representación general del escenario creado en Coppelia

El procedimiento general del algoritmo comenzará haciendo una foto RGB/ profundidad de la caja llena de objetos. La cámara de profundidad preprocesará los puntos, siendo elegibles sólo aquellos con una altura superior a un umbral, para evitar que el robot elija un punto en la base de la caja. A continuación, el algoritmo de aprendizaje por refuerzo seleccionará, tras la exploración/explotación, el punto que será elegido. La siguiente figura muestra un esquema de la CNN y de la Red Neural, que relaciona la instantánea de la cámara con la variable éxito/fracaso. Nótese que la capa final de la NN es un solo nodo (éxito/fracaso).

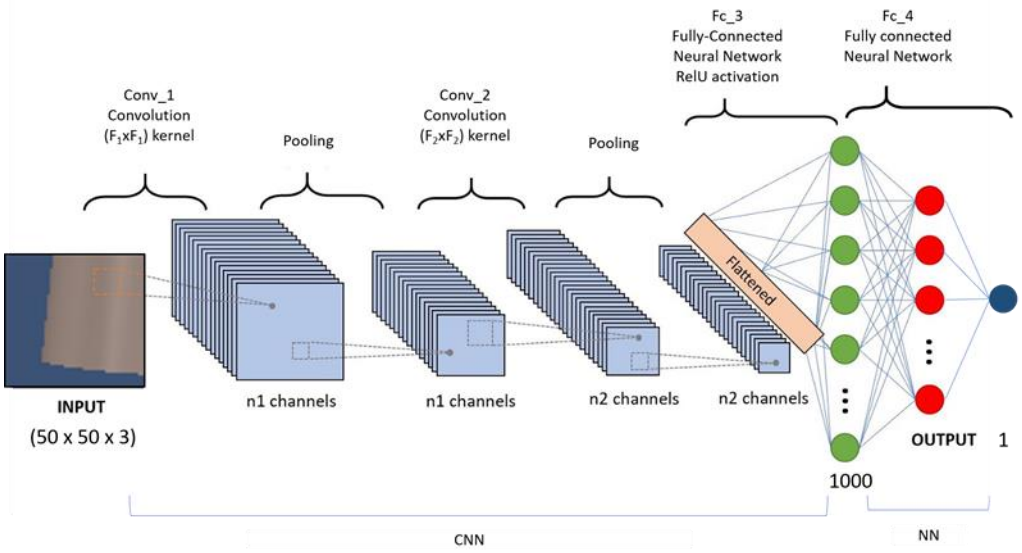

Figure 2 Representación general del algoritmo CNN+NN construido

Una vez seleccionado el punto, el robot coloca la ventosa en el punto y la activa. Si se ha cogido una pieza, el robot coloca el objeto en la otra caja. Si no, vuelve a la posición de seguridad (la de la figura anterior). A medida que el algoritmo ha pasado por más episodios, el algoritmo de aprendizaje por refuerzo utilizará predominantemente las redes neuronales para seleccionar los puntos, y la precisión del algoritmo será cada vez mayor.

Una vez desplegado el modelo, se realizará un análisis de sensibilidad para encontrar los hiperparámetros óptimos que proporcionen las mejores precisiones. Además, se mostrarán dos ideas de mejora en la creación del batch de imágenes utilizado para

entrenar la NN, que generalmente es aleatorio. La primera consistirá en seleccionar aquellos puntos cuya predicción se acerque más a 0,5 (muestreo de los indefinidos, el algoritmo no decide entre éxito/fracaso). La segunda seleccionará aquellos puntos cuya predicción esté más alejada del resultado real del punto (muestreo de los peor predichos).

4. Resultados

La figura muestra el crecimiento de la precisión a lo largo de los episodios en el algoritmo de aprendizaje por refuerzo, para los hiperparámetros optimizados, para los tres casos mencionados anteriormente (muestreo aleatorio, muestreo de los indefinidos y muestreo de los peor predichos, respectivamente).

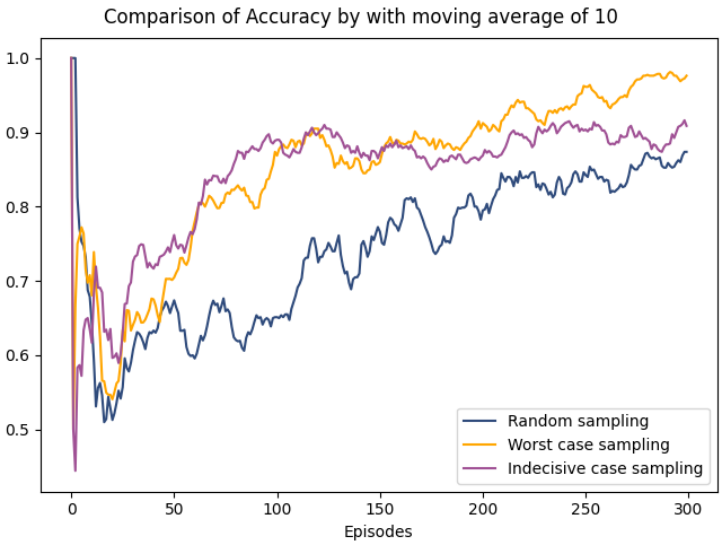

Figure 3 Comparison on accuracy on the reinforcement learning algorithm for several algorithms

5. Conclusiones

Hay varias conclusiones que se desprenden de los resultados. La primera conclusión es que el muestreo de los indefinidos parece tener una mejora más rápida de la precisión, pero una precisión final inferior a la del muestreo del peor caso. La segunda es que la mejor solución es el muestreo de los peor predichos, porque la precisión final es muy superior a la de las otras dos soluciones. La precisión final a lo largo de 300 episodios de entrenamiento se sitúa en torno al 97%.

El proyecto muestra resultados prometedores en un proyecto de producción. No obstante, el siguiente paso principal debería ser desplegar la solución en el laboratorio real, para confirmar los resultados en un escenario físico.

Otro proyecto futuro interesante es mejorar el gemelo digital (es decir, mejorar el vacío) para dar una representación más realista de la realidad, o crear un sistema que empiece a trabajar en el gemelo digital y luego con el entorno real. De este modo, la red neuronal empezaría a trabajar en el entorno real con pesos preentrenados y, por tanto, la convergencia sería más rápida.

Reinforcement learning applied to a Universal Robot with a vacuum gripper

Author: Burgos Madrigal, Álvaro.

Director: Juhel, Phillipe.

Entidad Colaboradora: ICAM Toulouse

Key Words: *vacuum gripper, Resnet, CNN, NN, Replay Memory, Reinforcement Learning*

1. Introduction

It is unquestionable that artificial intelligence is one of the key challenges of society this era. Artificial intelligence has proven to be suitable for a wide variety of use cases, from helping in Big Data targeted decisions (i.e. targeted marketing in social networks), to provide a robot the ability of walking, speaking, or even playing games, getting to even better levels than humans.

This intelligence, among the rest of technologies that conform the smart industry, is designed with two main objectives. The first objective is to improve the efficiency and efficacity of the current industrial processes. When a robot learns to do a task by his own, he will be often capable of delivering the same result as a human but in less time, with fewer defects and consuming less resources. These improvements will drive long term rentability on the enterprises and thus it is destined to arrive. The second important objective of the smart industry, and specially in reinforcement learning, is to place the humanity where the true value of the enterprise is. A robot will mostly serve to eliminate the repetitive tasks that a worker is forced to do and that do not contribute to giving meaning to his life, and to assist the human providing enhanced visibility where the human skills don't keep up.

The purpose of this project follows these objectives. The project and report that will be presented represent the next steps of a Research & Development team of the university of ICAM, in Toulouse. The team started developing this project two years ago, and its final objective is to teach a robot to pick up objects using vision. The scenario which the robot is set to resolve is formed by two boxes, one camera and the robot itself. One of the boxes is full of the same objects, and the other one is empty. Then, the objective is that the robot detects the point with the highest probability of being apt for the pick. Then, the robot will then place the vacuum gripper in the point selected and successfully pick the object and place it in the other box.

The main complexity of the problem is due to the lack of information about the object that is going to be picked. The algorithm will only have access to a color camera, a depth camera, and will know the depth of the floor.

This project, could ultimately have various applications. Among them, this robot, trained correctly, would be able to place some objects, such as components of a product, screws, tools etc, from the inventory of a factory to the plate of an AGV which could, once the robot has finished approach these items to the operator, save him the time and the discomfort of having to stand up to go from the workspace to the storage and carry all the needed items with himself.

2. Definition of the project

The prior version of the solution uses a supervised learning algorithm to train the robot, based on three phases. The first one is the data acquisition, in which the robot tries to pick randomly points inside the box full of objects and saves a cropped snapshot of the camera around the selected point with a tag of success/fail depending if the object was picked or not. The second one is the training phase in which a Residual Convolutional Neural Network (Resnet CNN) and a Neural Network (NN) are used to provide the optimal weights to predict success/fail with the snapshot of the point selected. The Convolutional Neural Network is already trained to provide the main characteristics of a snapshot. The third one is the test phase, which ultimately will be the production phase, in which the algorithm is tested to calculate the accuracy.

Nevertheless, the project explained in this report will explore a different solution, using reinforcement learning. The CNN will still be a Resnet pretrained, but the NN will follow a reinforcement learning algorithm, which will convert the three phases mentioned above into one dynamic algorithm.

Additionally, one of the pain points of the researches over the years have been dealing with the problems of the robot, not the CNN/NN algorithm itself. Thus, the project will be built under a digital twin of the laboratory in Toulouse, using the simulation software Coppelia.

To do so, the 6 months of the project will be divided in three parts equally distributed in time. The first two months of the project will focus on acquiring the expertise on both the reinforcement learning and Coppelia. The second two months of the project will be reserved for developing the software and the algorithm. Lastly, in the last two months the results will be analyzed, a sensitivity analysis will be conducted, and the report will be written.

3. Description of the model

The following figure shows the scenario built in the software Coppelia. The robot UR5 is in the middle of the scenario, and has two boxes at either sides of the robot, one full of objects and the other one empty. Each box has two different cameras, one RGB and one that measures depth. These cameras can be seen on the top left and top right of the figure.

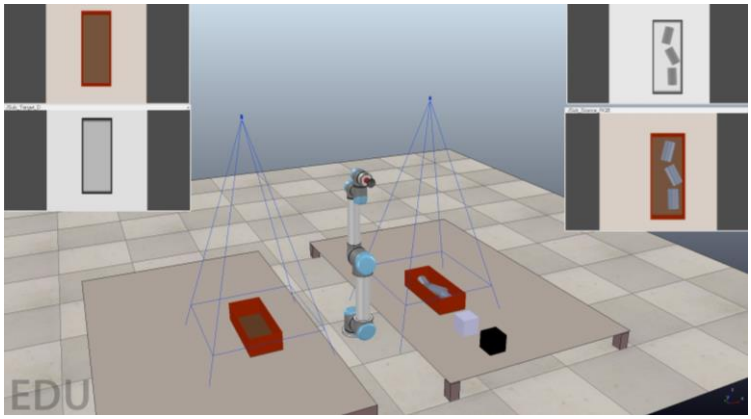

Figure 4 General overview of the enviroment Coppelia

The general procedure of the algorithm will start by making a photo RGB/Depth of the box full of objects. The depth camera will preprocess the points, being eligible points only those with a height higher than a threshold, to prevent the robot from picking a point in the base of the box. Then, the reinforcement learning algorithm will select, following exploration/exploitation, the point that will be picked. The following figure shows a schema of the CNN and the Neural Network, that relates the snapshot of the camera with the success/fail variable. Note that the final layer of the NN is only one node (success/fail).

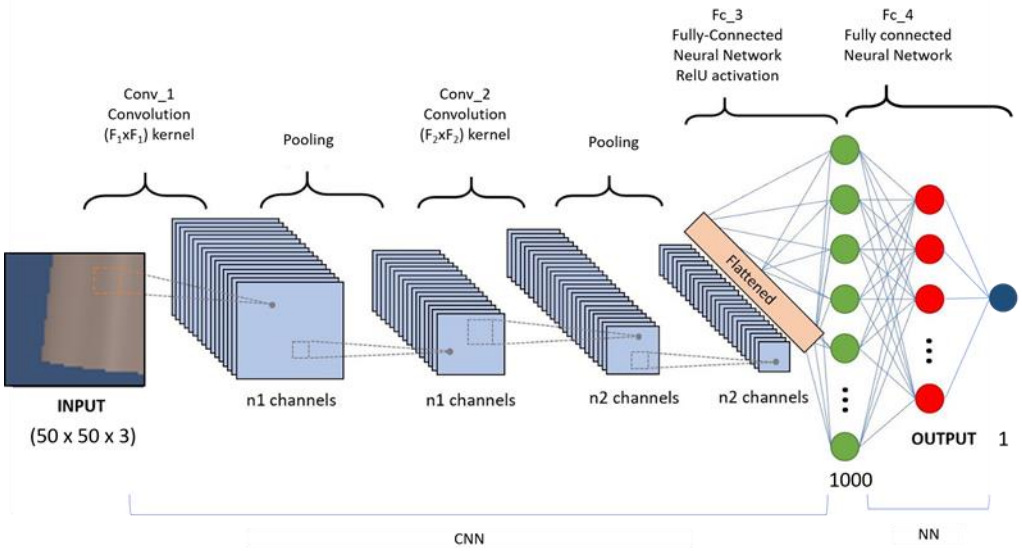

Figure 5 General Overview of the CNN+NN algorithm deployed

Once the point has been selected, the robot targets the point, places the vacuum gripper there, and activates it. If the pick was successful, the robot places the point in the other box. If not, it returns to the position of safety (the one of the previous figure). As the algorithm improves, the reinforcement learning algorithm will predominantly use the Neural Networks to select the points, and the accuracy of the algorithm will be higher and higher.

Once the model is deployed, a sensitivity analysis will be conducted to find the optimal hyperparameters that provide the best accuracies. Additionally, an improvement on the batch of images used to train the NN will be presented. In a general reinforcement learning algorithm, the batch of images is randomly selected. In this project two new ways of selecting the batch of images will be presented. The first one will be selecting those points whose prediction is closer to 0,5 (algorithm not deciding between success/fail). The second one will select those points whose prediction is further from the real outcome of the point.

4. Results

In a nutshell, the following figure shows the accuracy growth over episodes in the reinforcement learning algorithm, for the optimized hyperparameters, for the three cases mentioned above (Randomly Sampling, Indecisive Sampling and Worst-case Sampling, respectively).

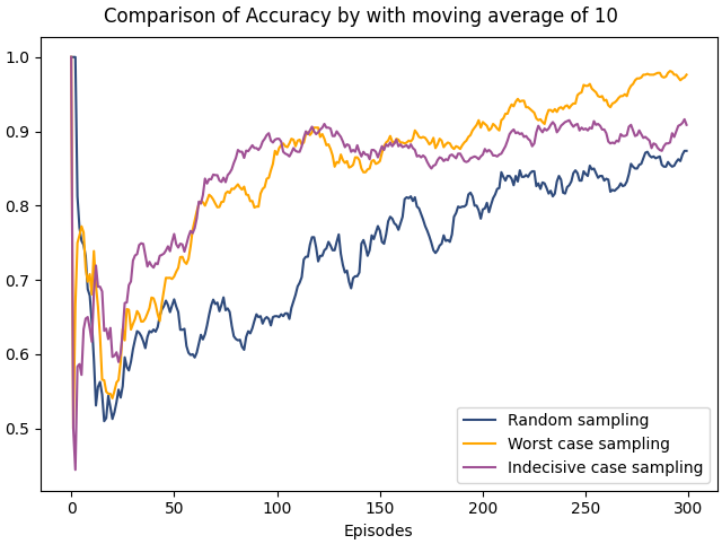

Figure 6 Comparison on accuracy on the reinforcement learning algorithm for several algorithms

5. Conclusions

There are several conclusions to drive from the results. The first conclusion is that the indecisive sampling seems to have a faster improvement in accuracy, but a lower final accuracy than the worst-case sampling. The second is that the best solution is the worst-case Sampling, because the final accuracy is far higher than the other two solutions. The final accuracy over 300 episodes of training is around 97%.

The project shows promising results in a production project. Nevertheless, the main next step should be to deploy the solution in the real laboratory, to confirm the results in a physical scenario.

Another interesting future project is to improve the digital twin (i.e. improving the vacuum gripper) to give a more realistic representation of reality, or to create a system that starts working on the digital twin, and then with the real environment. By doing this, the Neural Network would start working in the real environment with pretrained weights and thus the convergence would be faster.

UNIVERSIDAD PONTIFICIA COMILLAS
Escuela Técnica Superior de Ingeniería (ICAI)
Máster en Ingeniería en Tecnologías Industriales

Table of contents

Table of contents

Table of contents

Index of figures

Index of figures

Index of figures

Chapter 1. INTRODUCTION

1.1 MOTIVATION OF THE PROJECT

It is unquestionable that artificial intelligence is one of the key challenges of society this era. Artificial intelligence has proven to be suitable for a wide variety of use cases, from helping in Big Data targeted decisions (i.e. targeted marketing in social networks), to provide a robot the ability of walking, speaking, or even playing games, getting to even better levels than humans.

But the artificial intelligence is destined to be more transcendental. It is one of the angular pieces of the industry x4.0, which will disrupt the present paradigm of the day-to-day industrial processes as they are known.

This intelligence, among the rest of technologies that conform the smart industry, is design with two main objectives. The first objective is to improve the efficiency and efficacity of the current industrial processes. When a robot learns to do a task by his own, he will be often capable of delivering the same result as a human but in less time, with fewer defects and consuming less resources. These improvements will drive long term rentability on the enterprises and thus it is destined to arrive. The second important objective of the smart industry, and specially in reinforcement learning, is to place the humanity where the true value of the enterprise is. A robot will mostly serve to eliminate the repetitive tasks that a worker is forced to do and that do not contribute to giving meaning to his life, and to assist the human providing enhanced visibility where the human skills don’t keep up.

1.2 INTRODUCTION TO THE PROJECT

The project started two years ago, and its sole final objective is to teach a robot to pick up objects using vision. The scenario which the robot is set to resolve is formed by two boxes, one camera and the robot itself. One of the boxes is full of the same objects, and the other one is empty. Then, the objective is that the robot detects the point with the highest probability of being apt for the pick. Then, the robot will then place the vacuum gripper in the point selected and successfully pick the object and place it in the other box.

Afterwards, as they are still objects in the box, the robot will continue picking up objects and placing them in the other box using the same process until it is empty. Once this is fulfilled, the robot will understand that now the empty box is the one that was previously full and the full the one which was previously empty, thus resetting the scenario and training again.

The main complexity of the problem is due to the lack of information about the object that is going to be picked. The algorithm will only use a color camera, a depth camera, and will now the depth of the floor.

This project, could ultimately have various applications. Among them, this robot, trained correctly, would be able to place some objects, such as components of a product, screws, tools etc, from the inventory of a factory to the plate of an AGV which could, once the robot has finished approach these items to the operator, save him the time and the discomfort of having to stand up to go from the workspace to the storage and carry all the needed items with himself.

Additionally, these items could be heavy, and thus the presence of robots will eliminate the need of lifting weight. However, in the current prototype of the project this possibility is discarded as the robot used is the Universal Robot 3, which can lift a maximum of three kilograms.

Chapter 2. OVERVIEW OF THE TECHNOLOGIES

2.1 MACHINE LEARNING OVERVIEW

Machine Learning is defined by IBM as a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. All machine learning problems have in common that little to no information is known for the problem given beforehand.

In fact, in some cases, giving preprocessed information to a model can drive biases in the results, such limiting the outcomes to a subset of all the possible results, or even giving a wrong answer. For instance, an algorithm sharing the purpose of the project's algorithm could be trained to detect zones where a vacuum gripper can grip a plastic bottle of Coca-cola using a camera. If the algorithm was set to only look for the red pixels (as the label is mostly red), the robot would be having a higher accuracy in the general cases. Nevertheless, its solution would be limited to the red pixels and thus when an obstacle hides the label or the label is old, the algorithm would have poor results. Additionally, prefeeding the algorithm with information about the environment drives problems of scalability. In this example, the algorithm would be ready to work with Coca-cola bottles, but it will not work to pick bottles of Sprite.

Another common characteristic of a machine learning problem is that there are fed with datasets that are formed by a set of instances, organized in a set of features. Following the example of Figure 7, the species of a flower could be predicted by analyzing the dimensions of its sepals' petals.

The features would be each name in the header unless the species, each instance will be each row and the dataset would be the group of instances that will be used for developing the algorithm.

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
6.85	5.01	2.34	2.21	versicolor
6.77	4.98	2.41	2.29	setosa
2.28	2.15	2.76	2.65	virginica
1.43	0.93	1.21	0.43	virginica
6.77	4.98	2.41	2.29	versicolor
5.07	4.23	4.15	3.71	virginica
6.38	4.83	2.8	2.69	versicolor
1.82	1.55	1.77	1.49	setosa
6.3	4.8	2.89	2.77	setosa
3.64	3.37	4.61	3.98	versicolor
5.67	4.52	3.57	3.31	versicolor
5.47	4.43	3.77	3.46	versicolor
7.84	5.36	1.49	1.04	virginica
7.92	5.39	1.43	0.94	setosa
4.54	3.94	4.53	3.94	versicolor
4.95	4.17	4.25	3.77	versicolor
5.76	4.56	3.47	3.24	versicolor
2.62	2.51	3.43	3.21	versicolor
3.92	3.56	4.69	4.02	virginica

Header
Dataset
Instance

Figure 7 Illustrative example of a Dataset

In this project machine learning will be applied to improve the decision-making process of the robot, when it comes to selecting the point where the robot will place its vacuum gripper. Thus, the machine learning algorithm will be fed with a snapshot of the 50 px around every point where the robot has ever tried to go, and outcome 1 if the pick was successful or 0 if it was a failure.

2.1.1 Types of Machine Learning Algorithms

2.1.1.1 Supervised Learning

In these problems the objective is to predict one of the features, called dependent variable, by using the rest, called independent variables. Therefore, once the model is deployed, the algorithm will be able to access to all the independent variables and will make a prediction of the dependent variable, before its true value is known.

The supervised learning can be divided into classification algorithms, which is set to predict a discrete value, and regression algorithms, who is set to predict a linear variable. For instance, in an e-commerce retail customer target algorithm, a classification algorithm may respond to “will the customer buy an item in this connection?” whereas the latter may predict a linear variable, such as “what is the amount of money that the customer will spend in this order?”.

2.1.2 Unsupervised Learning

Unsupervised learning does not predict any feature, but gives information that wasn’t available yet. There are three clear examples of unsupervised learning, which are the following:

1. Clustering: This algorithm will create groups of data given a set of features. For instance, a retail enterprise could be interested in an algorithm for clustering its customers into several categories, and create an ad-hoc offers and discount for each customer depending on the necessities and priorities of each cluster.
2. Principal Component Analysis: In a machine learning algorithm that contains large sets of features (hundreds – thousands), a PCA is useful to reduce the number of features and create relevant, orthogonal features that can simplify the computation of the principal ML algorithm.

2.1.3 REINFORCED LEARNING

Reinforcement Learning is a subfield of Machine Learning, but is also a general-purpose formalism for automated decision-making and AI. A reinforced learning algorithm benefits from a continuous experience on the environment, enlarging the available dataset and continuously learning from the mistakes to improve recurrently the overall accuracy.

One of the bigger differences between a reinforcement learning algorithm and a supervised algorithm is the existence of consequent decision-making processes that depend on each other. In a classic reinforcement learning system the algorithm will interact with a scenario, starting from an initial position and making a decision based on the scenario. This decision will drive a new position in which the algorithm will have to make a decision. This process will be repeated until the scenario gets to an end and the scenario is reset, starting from the starting position again. The group of consequent decisions and positions until the scenario finishes is called an episode.

The process of continuously learning is defined using the concepts of state, action and reward, present in Figure 8. Firstly, a state gives the algorithm a full overview of the current scenario of the robot. Secondly, the algorithm chooses, from all the possible actions, one action. Thirdly, the action implemented in the scenario results in a reward and the next state. With this information in mind, the reinforced learning algorithm will learn to understand, for a given state, which actions are the ones that will maximize the total reward on the episode, which not only includes the reward resulted after an action, but also all the predicted rewards after the consequent actions on the states resulting from that action.

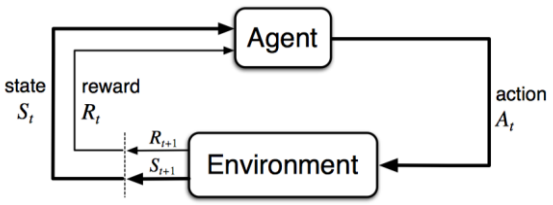

Figure 8 Reinforcement Learning Block Diagram

A visual example of a classic reinforcement learning algorithm is the Cartpole. In this scenario, there is a cart that can go to the left or the right and a bar attached to the cart in one inside and to a weight in the other one. In the initial position, the bar is in vertical position,

with the weight on the top. The objective is to keep the bar above the horizontal at every moment, by moving the cart to the right or to the left. The scenario can be seen in Figure 9.

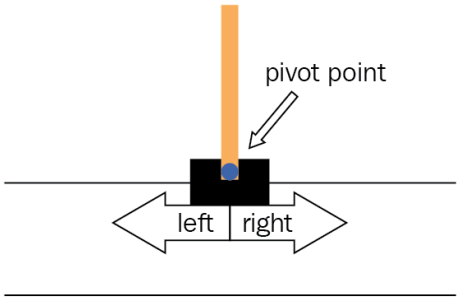

Figure 9 Cartpole Scenario illustration

In this problem statement, the state of the algorithm is given by two photos superposed with a lag between them, to capture the velocity of the cart and bar. The possible actions are going right or going left, at each iteration.

With respect to this project's algorithm, even though it will be based on reinforced learning algorithm, it will not follow exactly the continuous learning process described above, as there are several differences between a common reinforcement learning environment and this one.

The first difference is that every action the robot will take is independent from the rest of actions. The fact that the robot has effectively picked an object does not define the state in which it will be afterwards. Thus, every episode lasts only one state and one action, and thus it makes no sense for the algorithm to receive the next state at every decision.

Additionally, the definition of states and actions are not suitable for a reinforcement learning algorithm. Firstly, the concept of state is not well defined as the algorithm does not receive as input a state with a static format (i.e. a photo) but receives a list of all the possible points where the robot may land the vacuum gripper, defined by a snapshot of a 50px square around each point. Secondly, in a classical reinforcement learning algorithm there is an intrinsic lesson that the robot can learn from an action (in the cartpole, moving the cart to the right will create a positive angular acceleration on the pole, moving it to the left will create a negative angular acceleration). Nevertheless, there is not an intrinsic lesson that the robot can learn from selecting one point over the next one.

The algorithm is divided into several modules that interact with each other:

2.1.3.1 Environment

The environment is the scenario (system, machine, game, etc) where the user wants to excel. A reinforcement learning algorithm communicates with the environment in two steps. In the first one, the environment sends information about the current situation (current state, actions). In the second one, the environment receives the decision about the actions the

environment should do, and receive a reward for this action as a reply, as well as the next state of the environment.

Between the environment and the algorithm there is usually an environment manager, whose task is to mediate between the format the algorithm needs and the formats of the environment, for every state, action or reward.

In this project, the environment will be the virtual representation (made in Coppelia) of the laboratory in Toulouse. It will be described in detail in Chapter 5.

2.1.3.2 Agent

The agent is the decision maker who impersonates all the decisions that are made across the training of the algorithm. It is the agent that usually merges all the elements that form a reinforcement learning algorithm.

2.1.3.3 Replay Memory

At every timestep, the agent is in a state and has to make an action. The action leads to a reward and the next state. This tuple of state, action, reward and next step is called experience. Thus, Replay Memory is the term for the dataset that stores the agent's experiences while training, so they can be accessed for improving the accuracy of the results.

As the problem resolved in the project uses different concepts of states and actions, replay memory will be formed by tuples of all the points where the robot has tried to picked an object, represented by a preprocessed image of 50 px around the point of grip, and the reward of the action once executed.

2.1.3.4 Policy

The policy is the term that corresponds to the intelligence that recurrently progresses in the decision-making process across every training. In a classical reinforcement problem, the input is the current state of the environment, and the output is the recommendation of the action to be taken, having one node in the output layer for every possible action that the agent can take.

In the cases with some complexity, notably in those cases where the number of possible states is not defined (i.e. a state defined by a photo), a Neural Network is used to reduce the computation needed to achieve acceptable results.

In this project, as the concept of state is not clearly defined, there will be some modifications in the policy used. Firstly, the input of the Neural Network will be a preprocessed squared snapshot of 50 px centered in the point of grip. Thus, the neural network will have one node in the output layer, which represents the probability of the robot picking an object if positioning in this point.

Thus, for every episode of the training, in a classical reinforcement learning problem, the Neural Network is only used once, using the current state as input, and receiving an action

as output. However, in this project, for every episode the Neural Network will be run as many times as eligible points in the current scenario. Then, the agent will pick the point whose probability is the highest.

2.1.3.5 Strategy

The strategy stands for the Epsilon Greedy Strategy. In every reinforcement learning algorithm, there is always a balance between exploration and exploitation. If the agent is in mode exploration, it will pick an action randomly out of the possible options. Nevertheless, in the exploitation mode, the policy will be used to decide which of the actions is the most suitable, looking for maximizing the reward.

In the first episodes of the training, the epsilon greedy strategy must be programmed to explore predominantly, as the scenario is still unknown and an exploration phase could drive results biased. For example, in a scenario with several positive rewards, if only exploiting the environment, the agent could find the less positive reward first, and keep exploiting this reward, without adverting that there is a bigger reward at reach.

However, after the Neural Network is trained, if the scenario is not planned to change, the epsilon greedy strategy should be favoring more and more exploitation episodes, to maximize the output and perfectionate the current results.

2.2 NEURAL NETWORKS

Two types of neural networks will be used across the algorithm. The first neural network will serve to preprocess the snapshots received from the environment and convert them into a vector of length 1000, discretizing the essential information of the snapshot. To achieve this, a preprocessed Convolutional Neural Network (CNN) will be used, with pretrained weights that allow to differentiate between objects, animal, people, etc… As a consequence, the accuracy of the algorithm could be improved in a production phase by teaching the CNN to differentiate between the real images that are going to be tested, to create an ad-hoc mapping of snapshots to vector of length 1000.

The second neural network that will be used is a Traditional Neural Network. Its input will receive the output of the CNN and its output will be the probability of the robot picking the object (1 float number between 0 and 1).Figure 10 is an illustrative image of the two algorithms (unless the CNN is not Residual).

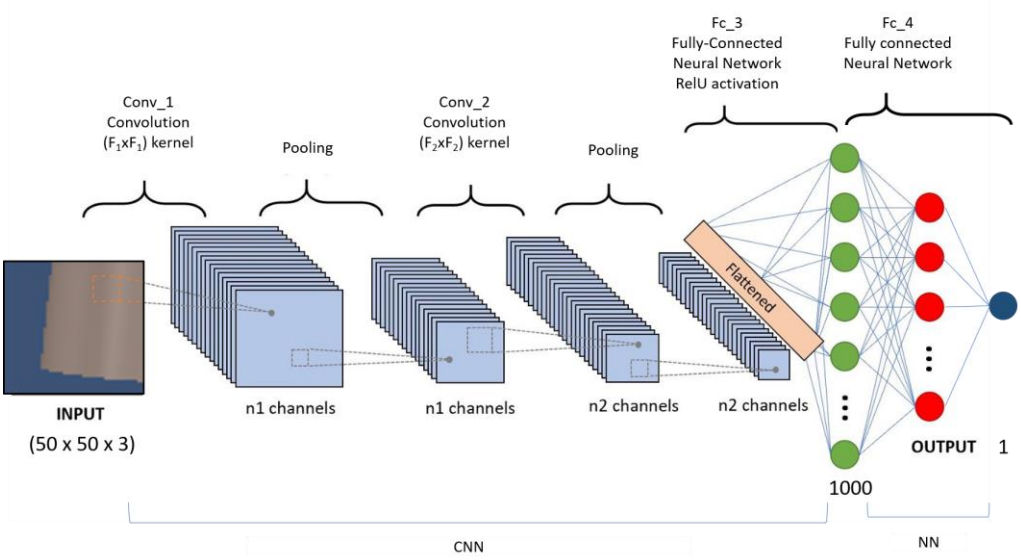

Figure 10 Illustrative Block Diagram of a CNN followed by a Neural Network

2.2.1 TRADITIONAL NEURAL NETWORK

A traditional Neural Network (NN) is comprised by a group of ordered layers, formed by nodes (or neurons). Each neuron is related to all the nodes of the previous and the following layer. The nodes of a layer are only a result of a multiplication of the inputs (previous layer nodes) by the weights with an added bias term, and an application of the activation function to the result. All these steps are essential to connect the input layer (which contains the input of the algorithm) with the output layer (contains the output of the algorithm) using nonlinearity relationships to solve complex results.

The main advantage of a Neural Network is that it is possible to have the greatest accuracies with little to no knowledge of the problem itself. Nevertheless, there are two main disadvantages to using Neural Networks. The first one is that the computation behind the algorithm is far higher than in other algorithms (such as random forests) and thus the computation time expected should be higher. Secondly, it is not possible to understand the calculations inside the network, and thus it is not possible to provide an explanation of the relationship between the feature and the output.

Before explaining in detail how a neural network typically works, it is appropriate to retrace a few steps to explain how a simple linear regression works, and subsequently, a logistic regression.

2.2.1.1 Linear Regression

A linear regression has the purpose of predicting a continuous variable from several inputs. As the prediction is linear, the equation is the following:

$$h_\Theta(x) = \theta^T x = \sum_{i=0}^{n} \theta_i x_i \tag{1}$$

With:

$h_\theta(x)$: The prediction of the dependent variable, y

θ_i: Weight of the variable x_i ($\theta_0$is the bias term)

x_i: Independent variable x. (x_0 is 1)

Therefore, we can define a loss function following the lasso formulation:

$$J(\Theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}{\theta_j}^2 \tag{2}$$

Note that there are two sums in this expression, that sum over different vectors. The former sum accounts for all the available experiences, whereas the latter accounts for all the elements in the vector of weights (except the first one).

This loss function should be minimized across every episode to provide better results. One of the most basic techniques that ensure minimization is the gradient descent algorithm, and although is has not been used in the project, it will allow to get a general idea of how the regression improves step by step.

With this objective, the gradient of the loss function is calculated, for every theta (which are the variables that will be optimized). For every step, the value of θ_i will be updated as:

$$\theta_0 = \theta_0 - \alpha\ \frac{\partial}{\partial\theta_0}J(\Theta) = \theta_0 - \alpha\ \frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})\ {x_0}^{(i)} \tag{3}$$

$$\theta_j = \theta_j - \alpha\ \frac{\partial}{\partial\theta_j}J(\Theta) = \theta_j - \alpha\ \frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})\ {x_j}^{(i)} + \frac{\lambda}{m}\theta_j \tag{4}$$

For $j \in [1, n]$ and α being the learning rate, and hyperparameter that will be optimized in the sensibility analysis. Therefore, in a linear regression, for every training, every theta parameter will be updated a fixed number of times, called epochs, minimizing the loss function at every epoch and thus increasing the overall accuracy of the algorithm.

2.2.1.2 Activation functions

Before diving into the classification problem, there is one major step that separates a linear regression problem from a classification problem, that should be explained apart, as it has also major implications in a Neural Network. This is the activation function.

The activation function is a function that lies between the output of the linear regression and the output, in a classification problem or in each node, for a Neural Network. It provides two main functionalities:

1. Determine the format of the output: Sometimes, the domain of the output is not the real numbers, and thus the image of the linear regression is not equal to the domain of the output. An activation function ensures that that these two vector spaces match. For example, in a classification problem, the output is always a number of real numbers between 0 and 1. (or a vector of real numbers). Another example, in a linear regression problem, is when a negative result does not have physical sense, such as predicting the average ticket of a customer entering in a web marketplace.
2. Add non-linearities to increase result complexity: Only nonlinear activation functions allow such networks to compute nontrivial problems using a small number of nodes. Otherwise the neural network may have the same output as a linear regression model.

Another important characteristic to consider when choosing an activation function is the saturation. A function is saturating if:

$$\lim_{|v|\to\infty} |\nabla g(v)| = 0 \tag{5}$$

When a function is saturating, it is said that they suffer from vanishing gradient, which usually drives slow convergence of the algorithm, as a big number of v does not change much in the output.

Some examples of non-linear activation functions are the following:

2.2.1.2.1 Sigmoid function/ logistic function

The sigmoid function is one of the most common functions to map predicted values to probabilities. It maps any real value into another value between 0 and 1. It follows the following equation:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

Graphically, the representation is the following:

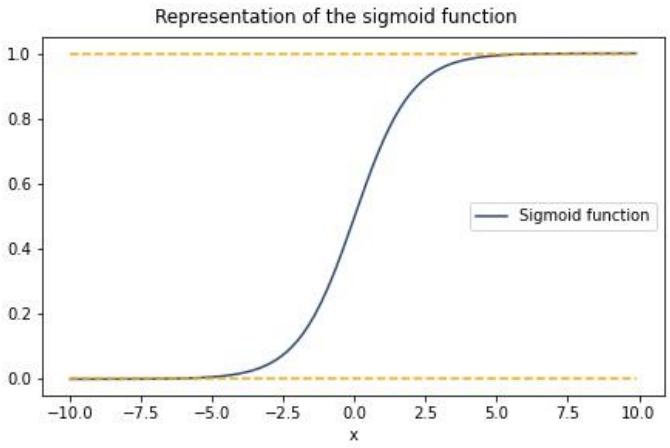

Figure 11 Sigmoid function graph representation

As it can be seen, there are two horizontal asymptotes in $y = 1$ and $y = 0$, for $x \in \mathbb{R}$. Additionally, the function is continuous and increasing monotonous, so the bigger x is, the closer to 1.

The advantages of using this function are that it is differentiable, so the slope can be computed in $\mathbb{R}$, and that it normalizes the output to 0 and 1. (for the result of a classification problem or for each node in a NN). Nevertheless, there are some disadvantages. The first one is that it has a vanishing gradient, thus it is prone to have a slow convergence. The second one is the function is computationally expensive, so it will probably require more computation time to converge.

2.2.1.2.2 Tan-h / Hyperbolic tangent

The hyperbolic tangent is an alternative to using the sigmoid function, as their behavior and shape is similar. The equation is the following:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{7}$$

Graphically, the representation is the following:

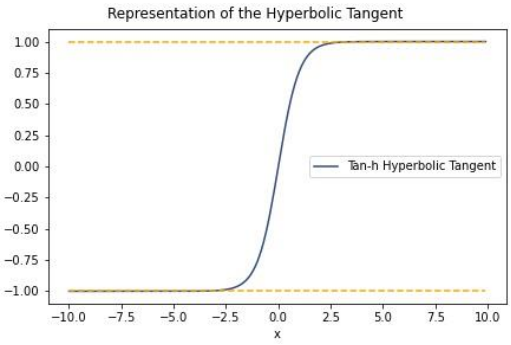

Figure 12 Tan-h/ Hyperbolic tangent function graph representation

Like the sigmoid function, the hyperbolic tangent has two horizontal asymptotes, in $y = 1$ and $y = -1$ for $x \in \mathbb{R}$.. These asymptotes could be $y = 0$ and $y = 1$ computing i.e. $g(x) = \frac{f(x)+1}{2}$, (note that then $g(x) = \frac{e^x}{e^x+e^{-x}} = \frac{1}{1+e^{-2x}} = s(2x)$, being s the sigmoid function). Likewise, the function is continuous and increasing monotonous.

The advantages of using this function are that it is differentiable, and that it is zero centered (useful in some applications where the input have strongly negative, neutral and positive values). As the sigmoid function, it also suffers from a slow convergence as it has the vanishing gradient problem.

As these two functions seem to be really similar, they should be compared to find the differences and suit each for each use case. One of the major differences of both functions is the behavior of their gradient. For the sigmoid function $s(x)$, and the tanh function $t(x)$, the gradients are:

$$s'(x) = s(x)\big(1 - s(x)\big) = \frac{e^{-x}}{1 + 2e^{-x} + e^{-2x}} \tag{8}$$

$$tanh'(x) = 1 - \ tanh^2(x) = \frac{4}{2 + e^{2x} + e^{-2x}} \tag{9}$$

Graphically represented, these are both functions:

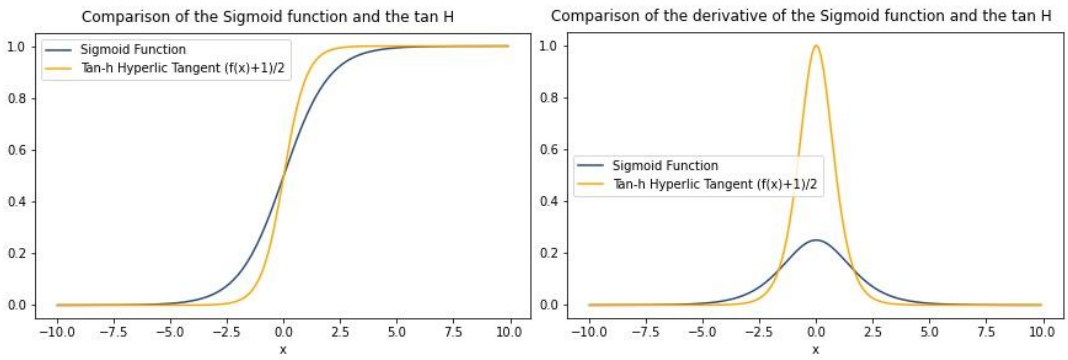

Figure 13 Comparison of the Sigmoid Function vs Hyperbolic tangent graphic representation

In the values next to zero, the gradient of the tanh is four times greater than the sigmoid function. Nevertheless, the dissipation of the functions, the sigmoid function is dissipating later.

When using activation functions in a NN, the data is usually centered around zero, especially in the first epochs, as the weights are initialized close to zero. Thus, at first sight, the tanh should be a better choice for a NN algorithm as it should converge sooner.

2.2.1.2.3 ReLU (Rectified Linear Unit)

This function saves the positive values and zeros the negative ones. The equation can be written as:

$$f(x) = \max(0, x) \tag{10}$$

Its function representation is the following:

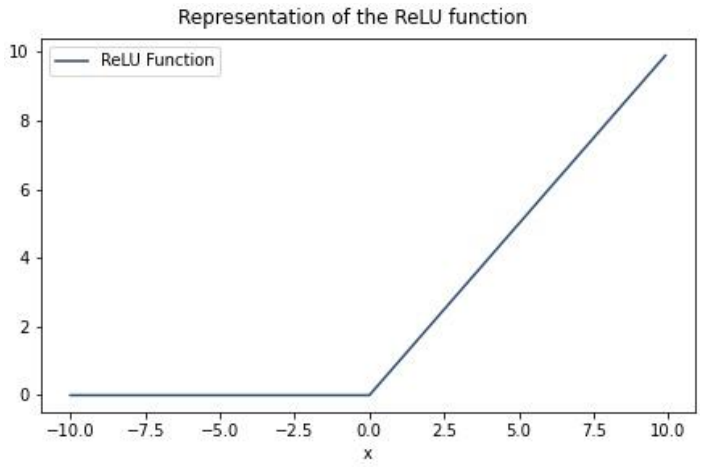

Figure 14 ReLU function graph representation

Note that this function is non-linear, even though it may not seem like it. Additionally, it is increasing monotonous (not strictly) and has a derivative in all real numbers but 0. The main advantage of this function is that it is computationally efficient, and thus allows the network to converge very quickly. For that reason, it will be the function used in this project, with a soft max in the final layer (it will be explained later). The main disadvantage of this function, is that the domain of where the function is activates is restricted to positive numbers, and thus the algorithm cannot learn when the input is lower than 0.

2.2.1.2.4 Parametric ReLU

This function is a modification of the previous function to solve the disadvantage commented on the function ReLU. The function is the following:

$$f(x) = \max(kx, x)$$

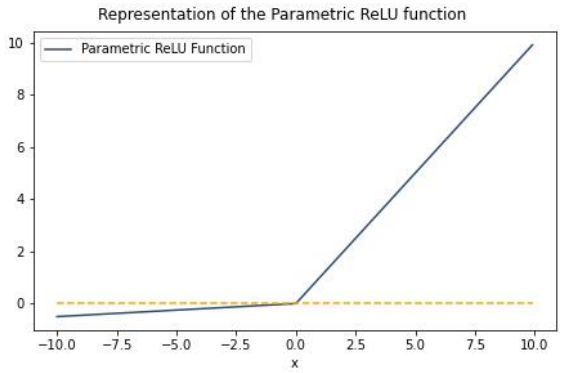

Figure 15 Parametric ReLU function graph representation

With the hyperparameter $k \in (0,1)$. Note that the ReLU function is the case for $k = 0$.

The advantage of this function is that prevents the ReLU problem, and the algorithm can improve when the input is lower than zero. Nevertheless, the results given for negative values are not consistent. Additionally, if the learning rate is high, the node is prone to overshoot and deactivate the neuron.

2.2.1.2.5 Softmax

This function is different that the function mentioned above as it takes into consideration all the results of the dataset, before outputting the result. It is a valid solution for computing probabilities, as it orders all the results from the most probable to be 1 (the highest) until the most probable to be 0 (the lowest). The equation is the following:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{m} e^{x_j}} \tag{11}$$

The main advantage of this function is that the normalization can be executed across all the instances of a dataset, for a univariate classification problem, or across all the classes in a multi-class classification problem. The former is useful when the algorithm will afterwards choose from all the instances the best or the worst solutions, as it orders the solutions in a probabilistic format. Likewise, a multi-class soft max normalization is useful when the algorithm will choose afterwards the class with a higher probability, as they are other for every instance across classes. For example, when deciding if a photo is an animal, a plant or a human, a softmax function across the classes will be a suitable choice.

Nevertheless, the main disadvantage is that this function can only be used for the output layer in Neural Networks and thus the rest of the nodes should be computed with other activation function.

2.2.1.3 Classification

However, in many cases the dependent variable is a discrete variable taking two possible values, which are 0 and 1. In this project, the Neural Network will respond to a classification problem representing the probability of the robot picking the object in the selected point.

As the output should be 0 or 1, the expressions presented in the linear regression are not valid, and an activation function needs to be used to force the output to be 0 or 1. Even though a ReLU and a softmax function have been used in the project, the sigmoid function will be used to explain the classification as it is more commonly picked.

When it comes to the loss function, it is not possible to replicate the cost function used in the linear regression as the square of the expression $(h_\Theta(x^{(i)}) - y^{(i)})^2$ results in a non-convex function. For this reason, the following loss function is defined:

$$Cost\ (h_\Theta(x), y) = \begin{cases} -\log(1 - h_\Theta(x)) & if\ y = 0 \\ -\log(h_\Theta(x)) & if\ y = 1 \end{cases} \tag{12}$$

Note that y can only be equal to o or 1. Written in one line, for every $x^{(i)}$:

$$J(\Theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log\left(h_\Theta(x^{(i)})\right) + (1 - y^{(i)}) \log(1 - h_\Theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n} {\theta_j}^2 \tag{13}$$

With $\frac{\lambda}{2m}\sum_{j=1}^{n} {\theta_j}^2$ being the regularization term. This function is suitable as it can be observed in Figure 16:

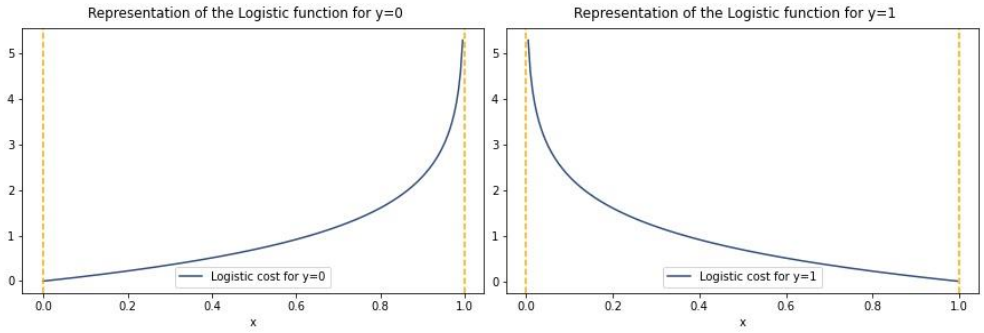

Figure 16 logistic cost in a logistic regression graph representation

As it can be seen, for y = 0, the loss function equals 0 when $h_\Theta\left(x^{(i)}\right)$ is close to 0 and tends to infinite when $h_\Theta\left(x^{(i)}\right)$ tends to 1. Likewise, for y = 1, the loss function equals 0 when $h_\Theta\left(x^{(i)}\right)$ is close to 1 and tends to infinite when $h_\Theta\left(x^{(i)}\right)$ tends to 0.

By following the same process in the gradient descent algorithm as in 2.2.1.1, the gradient of the loss function needs to be calculated, to update the weights (θ) at every step. The result is the same equation as the one showed in 2.2.1.1:

$$\theta_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\Theta) = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\Theta\left(x^{(i)}\right) - y^{(i)}\right) {x_0}^{(i)} \tag{14}$$

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\Theta) = \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\Theta\left(x^{(i)}\right) - y^{(i)}\right) {x_j}^{(i)} + \frac{\lambda}{m}\theta_j\right] \tag{15}$$

For $j \in [1, n]$ and α being the learning rate, and hyperparameter that will be optimized in the sensibility analysis. In every episode every weight will be updated a number of epochs to drive the loss function towards the absolute minimum of the function.

2.2.1.4 Nodes, Layers and Nomenclature in a Neural Network

A neural network is comprised in node layers, containing an input layer, an output layer, and one or more hidden layers (see image below). Each node of each layer is connected to all the nodes of the previous layer and all the nodes of the following layer, and therefore for each connection there exists a weight that will be optimized.

Every node has two elements called the hidden term and the activation term, and are expressed as $z_{(j)}^{(l)}$ and $a_{(j)}^{(l)}$ respectively, being l the number of the layer and j the node in the layer. Additionally, every relationship between two nodes is represented by a weight $\theta_{(i,j)}^{(l)}$,

being l the number of the layer, and i and j the output node and the input node of the regression, respectively. These relationships can be better illustrated in Figure 17.

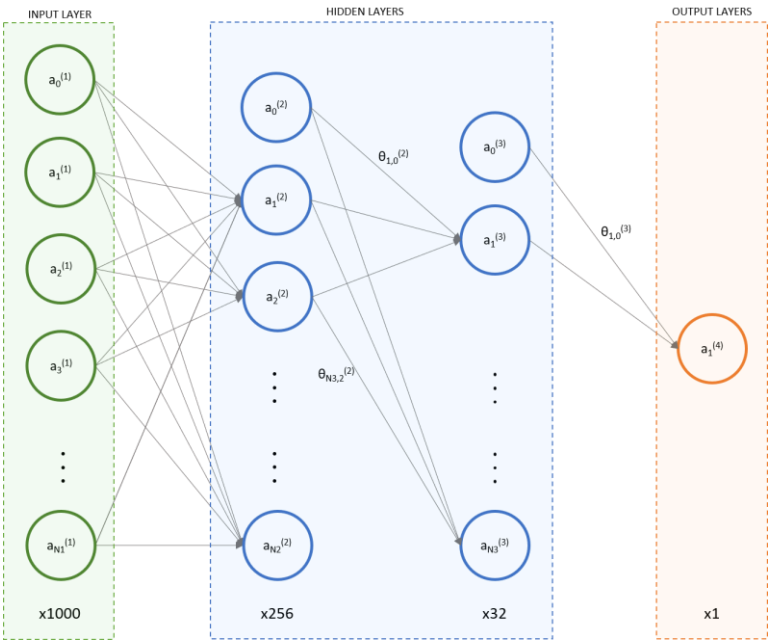

Figure 17 Neural Network visual representation for nomenclature porpuses

2.2.1.5 Forward and backward propagation

Once the concepts of layer and node are clear, the design of a neural network can be approached in a similar way to linear and logistic regression. In order to develop the algorithm, two expressions need to be computed: the loss function and the gradient of the loss function.

The function of loss of the Neural Network of this project can be easily formulated, as it can be seen as a complex logistic regression (the output is a Boolean variable):

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\Theta\left(x^{(i)}\right)\right) + (1 - y^{(i)}) \log(1 - h_\Theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{i}^{N_{l+1}}\sum_{j}^{N_l}(\theta_{i,j}^{(l)})^2 \quad (16)$$

Nevertheless, it is not yet defined how to compute $h_\Theta\left(x^{(i)}\right)$. Its computation is called forward propagation. Following the equations showed in the logistic regression, the forward propagation as:

$a^{(1)} = x$
$z^{(l)} = \Theta^{l-1} a^{(l-1)}$ for $l \in (1, L]$
$a^{(l)} = g\left(z^{(l)}\right)$ for $l \in (1, L]$
$h_\Theta(x) = a^{(L)} = g\left(z^{(L)}\right)$

Being $z^{(l)}, a^{(l)}$ vectors of dimension $N^{(l)}$, which is the number of nodes in layer l.

To compute the gradient of the loss function for each weight, the term propagation error $\delta_j^{(l)}$ is presented, which for every node j in layer l equals to:

$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} cost(i) \tag{17}$$
$$= \frac{\partial}{\partial z_j^{(l)}} \left(y^{(i)} \log\left(h_\Theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\Theta\left(x^{(i)}\right)\right)\right.$$

$$\delta^{(L)} = a^{(L)} - y \tag{18}$$

$$\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* g'\left(z^{(3)}\right) \text{ for } l \in (1, L) \tag{19}$$

Thus, it can be shown that the gradient of the loss function with respect to theta is:

$$\frac{\partial}{\partial \theta_{i,j}^{(l)}} J(\Theta) = D_{i,j}^{(l)} = \frac{1}{m} \sum_{k=1}^{m} a_j^{(l)} \delta_i^{(l+1)} + \lambda \theta_i^{(l+1)} \tag{20}$$

To conclude, in a Logistic Neural Network algorithm using a gradient descent algorithm, the steps to take in each epoch are the following:

1. Input: Training set $\left\{\left(x^{(i)}, y^{(i)}\right)\right\}$
2. Set $\left(\Delta_{i,j}^{(l)} = 0\right)$
3. For i=1 to m
   - Set $a^{(1)} = x^{(i)}$
   - Compute $a^{(l)}$ using the forward propagation
   - Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

- Compute $\delta^{(l)}$ using backward propagation
- $\Delta_{i,j}^{(l)} += a_j^l \delta_i^{l+1}$

4. Update θ:
   - if $j \neq 0$ $\theta_{i,j}^{(l)} = \theta_{i,j}^{(l)} - \alpha\left(\frac{1}{m}\Delta_{i,j}^{(l)} + \lambda\theta_{i,j}^{(l)}\right)$
   - if $j = 0$ $\theta_{i,j}^{(l)} = \theta_{i,j}^{(l)} - \frac{\alpha}{m}\Delta_{i,j}^{(l)}$

2.2.2 CONVOLUTIONAL NEURAL NETWORK

The Convolutional Neural Networks (CNN) are an area of deep learning that specializes in pattern recognition and extracting information out of them. It is widely used in images, even though it has other utilities.

The architecture of a CNN seems identical to the architecture of a NN: Both of them are constructed by nodes organized in layers. However, the main difference between those two is the computation of each node and the dimensions of it.

Firstly, in a NN each node is of dimension 1, whereas in a CNN a node is a tensor of X dimensions. In the input layer of the project, the input layer is formed by 1 node, representing a 50 px RGB photo, and thus is of size 3x50x50 (or 3 nodes of 50x50).

Secondly, there exists several transformations to pass from one node to the next one:

2.2.2.1 Filtering

It is the most characteristic transformation of the CNN, and is the responsible of detecting patterns across the image. The output node is the result of a convolutional product of the input tensor by a window matrix. When applying a filtering, the resultant tensor's dimension is reduced. The resulting dimension X given a filter of dimension F and an input of dimension N is $X = N - (F - 1)$.

Figure 18 introduces one of the most primitive and famous CNN problems, which intends to recognize hand-written numbers. The photo of the left is the original photo of one example of a seven. The photo of the right is the photo (black and white) converted into a matrix of pixels, and colored according to the value of pixels, being green the biggest (whitest) and red the lowest (blackest).

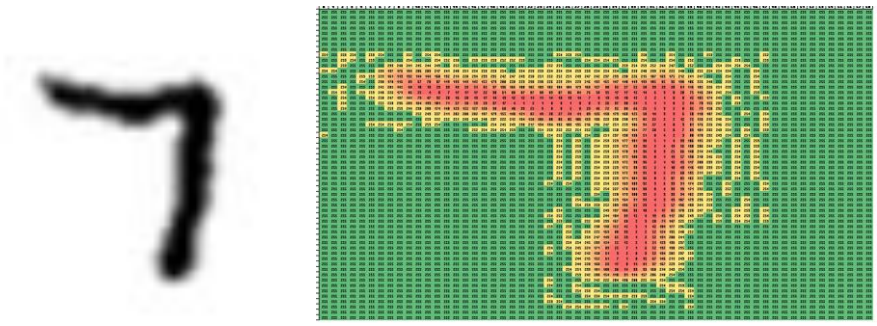

Figure 18 Original and pixel representation of hand-written seven

Figure 19 could represent a second layer of a CNN, having 4 nodes. Nevertheless, it has to be stated that this example is illustrative, and in a real CNN optimization, the nine weights of each transformation would be calculated by the algorithm, and thus are not that simple.

The two first figures represent two filters searching for vertical patterns. The filter matrix can be observed in the bottom left of each image. By looking at the left image, the greenest points correspond to those with white pixels on the left and black pixels on the right, thus defining the left contour in a vertical pattern. Likewise, the red points have black pixels on the left and white pixels on the right, having a negative and high output. This defines the right vertical contour of the number.

Additionally, as the matrices of the two images are opposites, the resulting nodes are also opposite, and thus the second image does not add any additional value in the optimization of parameters.

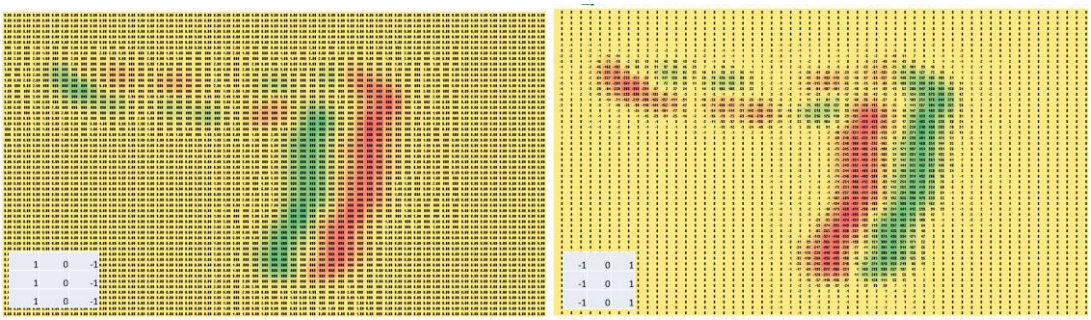

Figure 19 Vertical pattern recognition of a hand-written seven

Following the same logic, the left image below in Figure 20 represents the horizontal pattern, defining the upper contour of the seven the greenest points and the lower contour of the seven the red points.

The intention of the matrix filter on the right is to identify corners pointing upper right. By the same logic as the other two filters, the green points are representing the exterior contour of the matrix whereas the red represents the inner corner.

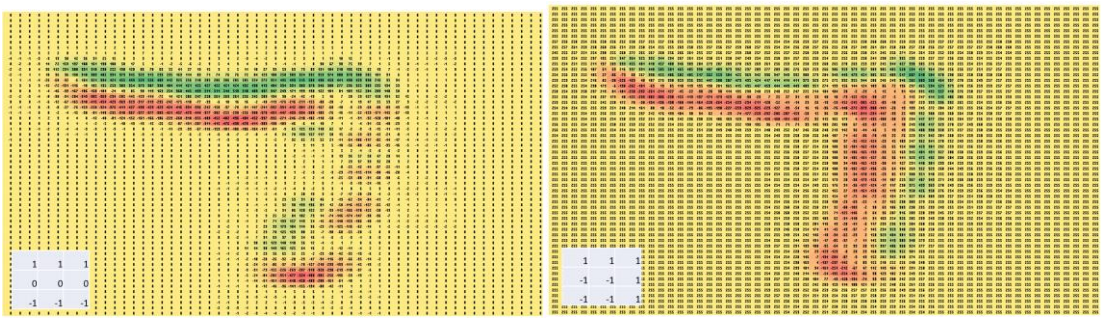

Figure 20 Horizontal and top right corner pattern recognition of a hand-written seven

2.2.2.2 Pooling

A pooling transformation reduces the dimensionality of the nodes. It groups the nodes by squares and computes an aggregation function, such as a max, min, sum, avg or median. The dimension of the output layer is $X = \frac{N}{S}$, for a Pooling of stride S.

For instance, Figure 21 represents two poolings of an input node 4x4 with a stride of dimension 2. The output node is of dimension 2x2. In the pooling on the top, aggregate function is the maximum of the four elements, whereas the pooling on the bottom computes the average of the group of elements.

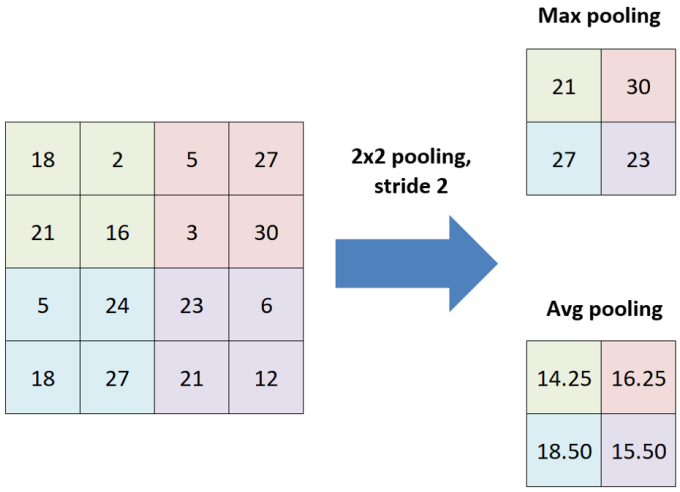

Figure 21 Illustrative example of a max pooling and an average pooling

In a convolutional matrix, the greater number of hidden layers the more sophisticated the results are. As a rule-of-thumb, with one convolutional layer, the algorithm may detect geometric shapes, such as edges, corners, squares or circles. With two convolutional layers the algorithm is probably able to recognize between more complex objects, such as eyes, fur, ears, hair or feather. With three convolutional layers (and beyond) the algorithm may differentiate even more sophisticated objects, such as cats, birds, humans or plants.

In this project a pre-trained Res-Net (Residual Network) of 50 layers will be used. The fundamental breakthrough of a Res-Net is the possibility of training deep neural networks with +150 layers. The main problem of a CNN with a high number of layers is the vanishing gradient problem, commented in 2.2.1.2. To reduce the decrease of the gradient of the value during backpropagation. Res-Net introduces a term called “Skip Connection”, which adds the original input to the output of the convolutional block.

As a 50 layer-Res-Net has a great number of hyperparameters and requires computation time and having many instances available, a pretrained Res-Net will be used instead. A pre-

trained CNN enables to transform the snapshots of a photo to values representative of the main characteristics of the photo. Although the meaning of each value of each output node is unknown, it is possible to train a NN after the CNN to predict the desired output.

2.2.3 Training, Cross-validation and Testing vs Exploration and Exploitation

As the training of a Neural Network has already been exposed, it is now important to distinguish the differences between the training, the cross-validation and test phase, characteristic of Supervised Learning, and the exploration and exploitation phases, characteristic of Reinforced Learning.

On the one hand, in Supervised Learning, the dataset is divided before training into the training subset (around 60%) the cross-validation subset (20%) and the test subset (20%). In the training phase, the weights of the algorithm (i.e. the weights of the NN) are calculated to minimize the loss, for a grid of hyperparameters. In the cross-validation phase, the predictions of the cross-validation subset are calculated, getting a loss value (or accuracy) for every point in the grid of hyperparameters. Then these loss values are compared and the optimal hyperparameters are selected. The result of the algorithm is the loss (or accuracy) of the predictions of the test subset. As a consequence of the results in each phase, and the curve of loss by number of iterations, one may find the balance between biased and over-fitted algorithms.

On the other hand, in Reinforced Learning, as it has been seen, the phases of exploitation and exploration are not consecutive. Instead the algorithm is learning and testing his improvement at the same time, and the goal is to maximize the overall accuracy of the algorithm. The main advantage of this approach is that the algorithm never stops learning, even in the production phase, as it is constantly updating the weights. As a result, the final accuracy of a reinforcement learning algorithm should be higher. However, there are some disadvantages. The first one is that the hyperparameters are usually fixed beforehand, in a sensibility analysis. The second one is that, if the algorithm is set to never stop learning, the computation resources needed in the production phase are much higher than in the Neural Network, where a forward propagation is enough for each state.

2.3 ROS

2.3.1 Overview Of ROS

Robot Operating System (ROS) is a set of software libraries and tools that help build robot applications, allowing to connect the sensors, the actuators and the control systems of a robot effortlessly.

To do so, ROS uses the concept of nodes, which is a process that performs computation. Nodes are combined together into a graph and communicate with one another using, among others streaming topics. For example, in the project there is one node managing each camera sending the photos taken, one node for the robot, sending information about the robot and

receiving orders of movement, and one node for the algorithm, which controls all the flows of information.

One of the main advantages of ROS is that programs can run in multiple computers and communicate over the network, allowing distributed computer systems. Additionally, it is multilingual, and thus it can be written in any language which a client library exist: C++, Python, Java, Matlab, Lua…

ROS currently supports TCP/IP-based and UDP-based message transport, being the former the default transport used and the only one supported by client libraries. The UDP connection separates messages into UDP packages and do not have acknowledgement, thus they are low-latency, lossy transports, whereas a TCP based message transport will be slower, ensuring the communication of the message.

2.3.2 Modes of Communication

Depending on the version of ROS (ROS1 or ROS2) the communication follows a master/slave architecture or is distributed. In the first case, the exchange of information is done via services, which are defined by a pair of message structures. The first message structure is for the request message, and the second one is for the respond. When one node has the intention of receiving a message from a service, sends a request message and awaits a response. The node that sends the information, which is already connected, receives the request message, performs the corresponding operation and sends back a response, which arrives at the first node, via the service. This can be shown in Figure 22:

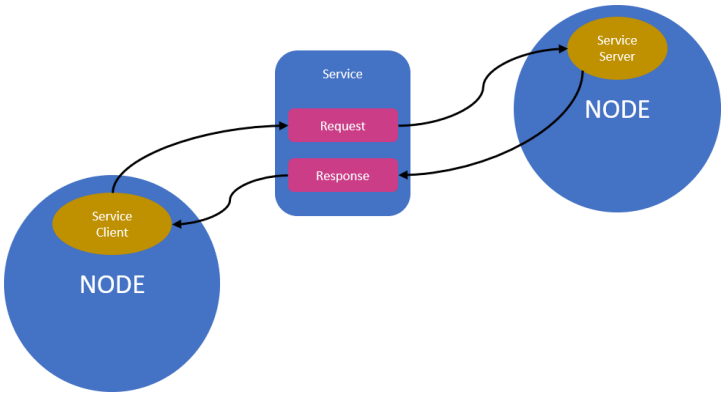

Figure 22 ROS Service Block Diagram

In the second case, a message passes from one node to the other through a topic. A topic is a named bus over which nodes exchange messages under a certain message structure. Nodes with the intent of sending messages will publish under the appropriate topic or topics, and nodes listening to the topic will have subscribed to the topic, specifying the correct message structure of the message. Note that topics allow the flow of information anonymously, and thus the node receiving information is not aware of who is receiving it. Likewise, they one who is sending the information is not aware of who is listening to it, nor if someone is even

listening, nor if there is more than one listener. Figure 23 represents several nodes communicating with each other:

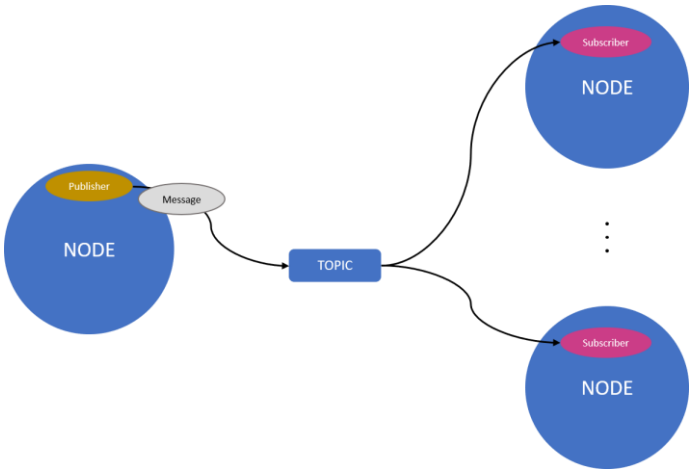

Figure 23 ROS Publish/Subscribe Block Representation

2.4 ROBOT

2.4.1 INTRODUCTION A ROBOT

The definition of a robot, by the ISO 8373:2012, is the following: An automatically controlled, reprogrammable, multipurpose manipulator programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications.

- Reprogrammable: designed so that the programmed motions or auxiliary functions can be changed without physical alteration.
- Multipurpose: capable of being adapted to a different application with physical alteration.
- Physical alteration: alteration of the mechanical system (the mechanical system does not include storage media, ROMs, etc.)
- Axis: direction used to specify the robot motion in a linear or rotary mode.

In order to explain how a robot moves, there are some concepts, similar for all the robots, that must be explained:

- Link: Each one of the rigid solid of the robot
- Joint: Place where two links are joint. Provides the Degrees of Freedom to the robot.
- Degrees of Freedom (DOF): Independent movements that one link can do with respect the previous one.
- Terminal Point: Point that represents the position of grip in the tool, where position is always the tuple position (x,y,z) and orientation (Euler angles)
- Wrist: Group of links and joints next to the tool (and terminal point) whose function is to orientate the robot the robot in the workspace of the tool.

- Reach/ Total Workspace: Volume on space where the robot can place the terminal point in. It can be limited by geometric factors (point far away), mechanical factors (links collide impeding access to the point)
- Singular Points: Points inside the reach where the robot struggles to place the terminal value. Often to get to those positions, little movement on the terminal value requires big movements of the joints.

2.4.2 KINEMATIC OF A ROBOT

Every link, as is a rigid solid, has one system of reference associated, to describe its movement. A system of reference is formed by three normalized, orthogonal, straight vectors, ordered so that they are dextrogeres (the third axis is the vector product of the first two axis). Every system of reference is defined by its axis and the point of reference, which defined the origin in its reference.

2.4.2.1 Matrixes of axis and points

The relationship between the axis of two systems of reference is the following:

$$\hat{X}_{s-1} = R\hat{X}_s = \begin{bmatrix} u_{ux} & u_{vx} & u_{wx} \\ u_{uy} & u_{vy} & u_{wy} \\ u_{uz} & u_{vz} & u_{wz} \end{bmatrix} \begin{bmatrix} u_u \\ u_v \\ u_w \end{bmatrix} \quad (21)$$

With $u_{ij} = \hat{u}_i \cdot \hat{u}_j$

For example, for a rotation in the z axis, matrix R is the following:

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad (22)$$

In terms of a point P, its position in the system of reference $(\hat{x}, \hat{y}, \hat{z})$ is $(P \cdot \hat{x}, P \cdot \hat{y}, P \cdot \hat{z})$, which is the orthogonal projection of the point in each axis.

In the following image, the point O is the origin of the system of reference s-1 (x, y, z), A is the origin of the system of reference s (u, v, w), and P is the point, with coordinates known in s, but unknown in s-1. Thus, P in coordinates s-1 (x, y, z) is:

(23)

$$P|_s = (P_u, P_v, P_w)|_s = A|_{s-1} + \left(\hat{X}_{s-1} \cdot \hat{X}_s^{\,T}\right) \cdot P|_s = P|_{s-1}$$

$$p_i = a_i + \hat{u}_i \cdot \hat{u}_u \cdot p_u + \hat{u}_i \cdot \hat{u}_v \cdot p_v + \hat{u}_i \cdot \hat{u}_w \cdot p_w \tag{24}$$

For $i \in (x, y, z) = s - 1$

This relationship can be shown matricially:

$$P|_{s-1} = \begin{bmatrix} u_{ux} & u_{vx} & u_{wx} & a_x \\ u_{uy} & u_{vy} & u_{wy} & a_y \\ u_{uz} & u_{vz} & u_{wz} & a_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_u \\ u_v \\ u_w \end{bmatrix} = {}^{s-1}|T|_s \cdot P|_s \tag{25}$$

Finally, for a robot of 6 DOF, the relationship of a point expressed in the global reference is:

$$P|_0 = {}^0|T|_1 \cdot {}^1|T|_2 \cdot {}^2|T|_3 \cdot {}^3|T|_4 \cdot {}^4|T|_5 \cdot {}^5|T|_6 \cdot P|_6$$

Where the variables of the movement are the variables that define the DOF, such as θ in (22). This relationship is called direct kinematics.

2.4.2.2 Euler Angles

Before explaining the inverse kinematics, the Euler angles have to be shown. As mentioned, every point in the space is defined by its position and its orientation, and every orientation can be defined by three rotations, in the z axis, in the x axis and in the z axis:

$$M = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$
$$= \begin{bmatrix} \mathrm{c}\,\theta\,\mathrm{c}\,\psi & -\mathrm{c}\,\phi\,\mathrm{s}\,\psi + \mathrm{s}\,\phi\,\mathrm{s}\,\theta\,\mathrm{c}\,\psi & \mathrm{s}\,\phi\,\mathrm{s}\,\psi + \mathrm{c}\,\phi\,\mathrm{s}\,\theta\,\mathrm{s}\,\psi \\ \mathrm{c}\,\theta\,\mathrm{s}\,\psi & \mathrm{c}\,\phi\,\mathrm{c}\,\psi + \mathrm{s}\,\phi\,\mathrm{s}\,\theta\,\mathrm{s}\,\psi & -\mathrm{s}\,\phi\,\mathrm{c}\,\psi + \mathrm{c}\,\phi\,\mathrm{s}\,\theta\,\mathrm{s}\,\psi \\ -\mathrm{s}\,\theta & \mathrm{s}\,\phi\,\mathrm{c}\,\theta & \mathrm{c}\,\phi\,\mathrm{c}\,\theta \end{bmatrix} \tag{26}$$

As a conclusion, the resultant matrix of rotation R of all the joints of a robot can be expressed in terms of the Euler angles, simplifying the calculation.

2.4.2.3 Direct and Inverse kinematics

With all the information presented above, it is possible to define now the direct and inverse kinematics of a robot, based on the information known and the information that wants to be known. The relationship can be shown in Figure 24:

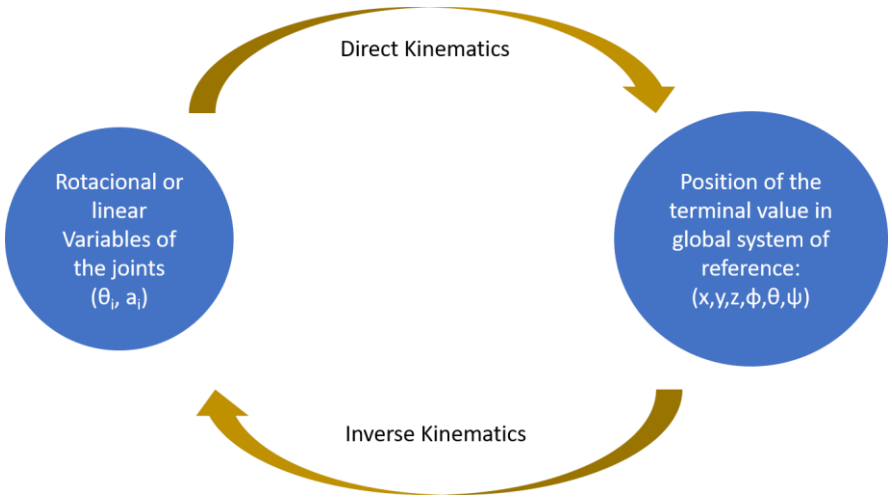

Figure 24 Direct and Inverse Kinematics inputs and outputs

Chapter 3. STATE OF THE ART

At the moment, the learning process of the algorithm is divided into three separated phases, that are launched separately. The first phase is the creation of the dataset, the second is the training and the third is the test phase.

1. **Dataset creation**

In this phase the algorithm is in its starting point and knows nothing about the environment but the snapshots of the box full of objects (color and depth) and the height of the floor. Thus, a grid of points inside the contour of the box is constructed, and filter by those with a height higher than the floor's (and a threshold).

The robot then chooses randomly a point out of all the possible points, and tries picking an object by placing the vacuum gripper there. If it successfully picks the object the snapshot taken of the point is tagged as success and if the object it is not taken it is marked as fail. This process is repeated until the dataset contains 400 success cases and 400 fail cases.

2. **Machine Learning Training**

Once the algorithm has acquired the correct number of snapshots with its outcome, the machine learning training starts. The snapshots are passed through a convolutional neural network (CNN from now on) to detect the relevant parts of the image and then by a Traditional Neural Network to predict the success or fail. The CNN used is the Resnet 18, which is tested to have the optimal balance between computational time and final accuracy.

The input of the Resnet is the RGB snapshot of the point with a size of 50x50, resulting in a tensor (n, 3,50,50), being n the number of photos at disposal. The output of the Resnet are 1000 points, which are not an output by itself, do not mean directly anything, but contain information about the whole snapshot.

The 1000 outputs of the pretrained CNN are the input of the classical neural network that is about to be trained. Currently the neural network used has 256 internal nodes in the first layer, 32 in the second one, and one final node in the output layer, to decide if the point is a success or is a fail.

The network is then trained by using the supervised learning approach of training/cross-validating. Finally, the weights of all the nodes are sufficient to classify new points as success or fail.

3. **Testing**

The algorithm implemented is currently tested with two different functionalities. The first one, called exploration, creates a mesh of points around the full boxes, and the algorithm calculates a probability of successfully picking an object in that point. With this information, by defining a 50% threshold, points are classified as success or fail, which can be confirmed or denied manually, by ordering the robot to find the outcome.

The second functionality plots all the possible points where the robot can go. The user selects a point, and the probability of the robot successfully picking the object is shown. If the user decides so, the robot can navigate though that point to try to catch the object, thus confirming or denying the hypothesis.

Even though it is clear that the project has already an Minimum Viable Product, there are some ideas that can be put in place to improve the overall performance of the system, which will be explored in this project.

Chapter 4. DEFINITION OF WORK

4.1 JUSTIFICATION OF WORK

The process presented above, although is operative in the laboratory, is not scalable to an enterprise product as it presents some issues:

1. Separated Phases: The three phases commented in Chapter 3. are completely different and a person is required to end and start some phases. The algorithm is not set to learn when there is a disposition of 400 snapshot of each category, and the third phase has to be launched afterwards.
2. Recursive learning: A problem that supervised learning has is that the results obtained cannot be studied to retrain and obtain a major accuracy. Thus, the model cannot perform any better once it is deployed, which could be done, considering that the number of tested images is increasing.
3. Dataset Size: To get one success/fail snapshot it takes a considerable amount of time as the robot, needs to move to pick the object, place it if it has picked it and return to the safety position. This process, accumulated over all the dataset is too time consuming.
4. Physical problems: Every physical system, such as the electronics, the vacuum gripper, the robot, the camera, are subject to changing their state and altering the results (adding bias) or breaking down, forcing to stop a training. (such as a transistor in the electronics burning).

In order to solve these issues, there are two main solutions that will be implemented in this project. The first solution is to create a digital twin of the laboratory, that will be targeting issues 3 and 4. The second solution will be to adapt the current state of the project to create a reinforcement learning algorithm, targeting issues 1, 2 and 3.

4.2 OBJECTIVES

The main objectives of the problem are the following:

- Explore the functionalities of Coppelia to find those who will best suit to create the digital twin of the model
- Learn about reinfocerment learning and build the structure of new system of the project
- Build the environment in Coppelia
- Build the new reinforcement learning algorithm basing the solution in the functionalities already built and deployed
- Perform a sensitivity analysis to find the optimal hyperparameters of the model

- Analyze the differences between the real and the virtual system and the possible differences in accuracy of both environments. Finding biases in the digital twin
- Analyze the results of the reinforcement learning problem applied to the digital twin and define the adaptations to be done so that it can be applied in the real environment
- Explore the next steps of the project

4.3 METHODOLOGY

This project started in march, and follows this timeline:

March	Follow a course on Reinforcement Learning
April	End march's course and explore Coppelia and ROS
May	Build the digital twin based on a simple environment and build simple connections ROS-Lua File with ROS-Python File
June	Reproduce the current algorithm CNN+NN to test and correct the environment behavior. Build the reinforcement learning algorithm and conduct the sensitivity analysis
July & August	Analyze the results, explore their implications and the next steps, and write the final reports and presentation of the project

4.4 OBSTACLES

Throughout the project, some obstacles have arised, which have greatly slowed the pace. Firstly, even though the laboratory and the project director were in Toulouse, the project was covered by remote in Luxembourg. Additionally, due to the circumstances, most of the project has been done in the weekends, where the laboratory is closed and the project director is not working. Finally, the reinforcement learning problem, such as all the machine learning problems, needs high computation resources, which are given by the one of the computers placed in the laboratory.

These facts have resulted in some obstacles such as:

- Connectivity Issues: There has been some problems of connection between the laboratory of the university and the computer in Luxembourg (computer turned off, wifi, to name a few) which have resulted in hours/days of inactivity.
- Availability: Even though all the parts of the project have been responsive through out all the project, the director was not available on the weekends and the student was almost not reachable in the working hours through the week, resulting in hours of inactivity when the project could not advance without the director's assistance.
- ROS: The relationship between ROS and Coppelia (configuration, version, etc) and ROS Master was launched in another computer from the laboratory. As a

consequence, ROS stopped working in some parts of the project pausing the development of the model Coppelia or the reinforcement learning algorithm.

Chapter 5. DEVELOPPED PROJECT

5.1 DIGITAL TWIN ANALYSIS

In this chapter the virtual environment created by Coppelia will be explained in detail to understand the similarities and differences with the real environment in the laboratory. The following picture shows the initial state of the environment every time the scenario is reset.

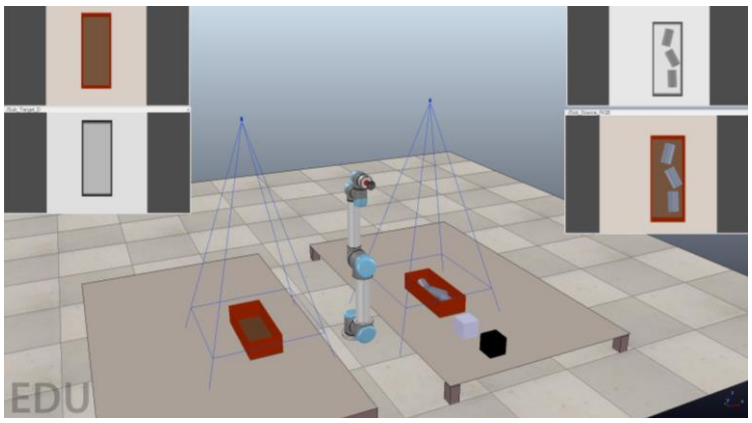

Figure 25 Coppelia Environment Representation

In the middle of Figure 25, the robot UR5 is shown in the position of safety. On the left and on the right of the robot there are the boxes of pick and place, being the full box the one for pick and the empty box the one for place.

The blue lines, on the left and on the right of the robot correspond to the limits of the camera. As it can be seen the camera does not works like an orthogonal camera, but a radial camera. There is one blue square above each box, which correspond to the minimal depth the camera detects. The maximal depth is aligned with the floor, and thus it cannot be seen.

On the upper left of Figure 25 two snapshots are shown, corresponding in this case to the box for placing objects. The upper snapshot corresponds to the RGB image and the lower snapshot correspond to the depth image, which is shown as a black and white image. In the upper right part of the image, similar snapshots are shown for the right box, which in this case corresponds to the box of pick.

5.1.1 ROBOT UR5

The robot UR5 which is used is an anthropomorphic robot, as its functionality is similar to the one of a human arm. The robot UR5 has 6 DOF and can lift up to 3 kg. As it can be seen in Figure 26, it is anchored to the ground and has a vacuum gripper as tool.

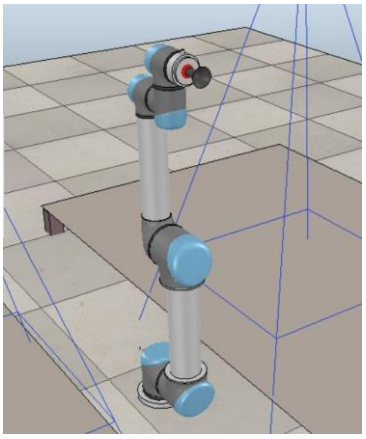

Figure 26 Digital twin of the robot UR

5.1.2 VACUUM GRIPPER

The model of the vacuum gripper created can be shown in Figure 27:

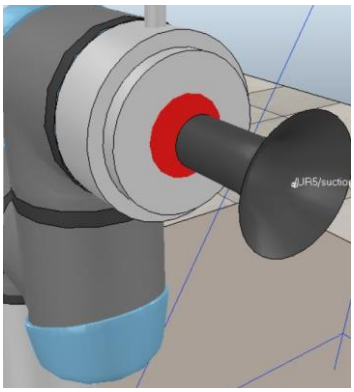

Figure 27 Digital representation of the vacuum gripper and the sensor to grip objects

A real vacuum is a tool that grippes objects by suctioning air and creating upper pression between the vacuum gripper and the object that is going to be gripped, hence attaching the object to the tool.

Nevertheless, the current model of the vacuum gripper is far simpler than this process. In the Figure 27 a point in the middle of the tool is shown. In this model, if the object touches this point, the gripper will successfully attach the object.

This may lead into some bias of the result and does the model should be improved as a next step of this project.

5.1.3 OBJECTS

The objects that are going to be picked are cylinders, as it done in the laboratory, and have similar dimensions. Figure 28 shows that the three objects are distributed randomly across the box:

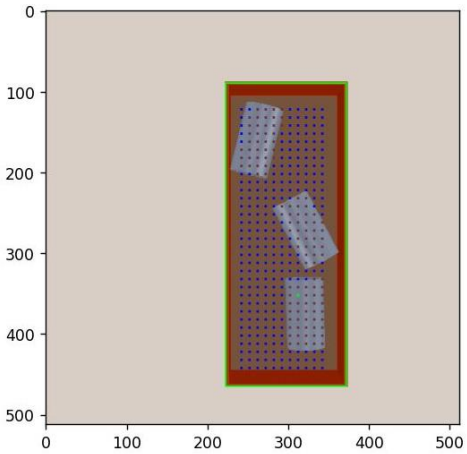

Figure 28 Preprocessed photo of the virtual box with contour, grid, and the point where the robot will pick

The preprocess induced in this image will be explained in 5.2.1. Once the robot has tried to pick the point, the experience is saved in the memory, by using a 50-pixels-snapshot, such as in Figure 29:

Figure 29 Snapshots of fail/success that will input the Resnet (50 px)

5.1.4 IMAGES

The images taken have a resolution of 512 pixels, for the depth and RGB pixels. A snapshot of all the cameras in the moment of pick is shown in Figure 30:

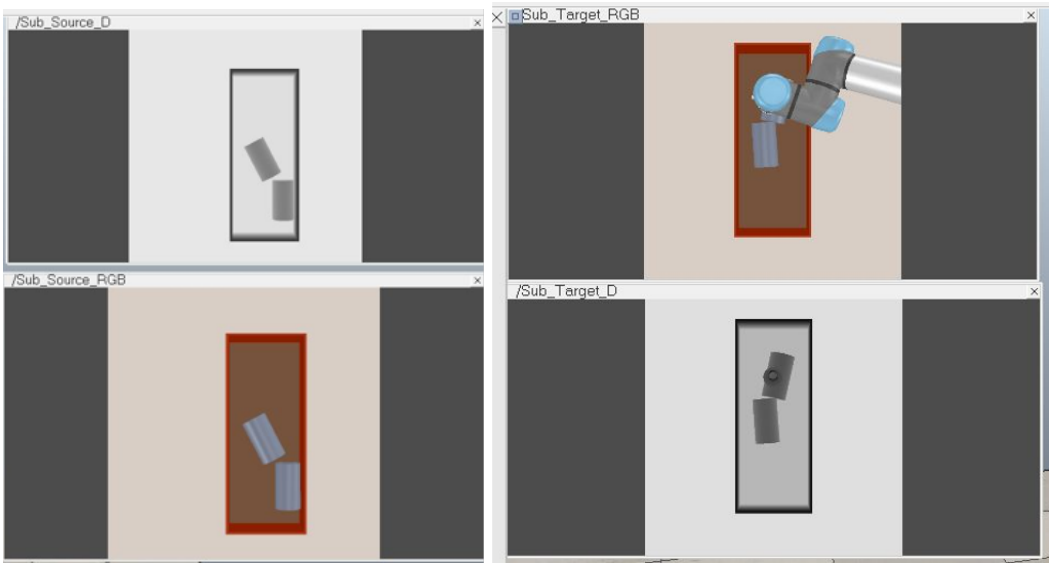

Figure 30 Photos depth/RGB of the left/right cameras in a random moment on training

On the place images (left), the depth and RGB cameras show that there are two objects that have been successfully placed in the place box. On the right side of the image, the robot is picking one the objects.

As the depth camera has a threshold, it does not detect the links and the joints of the robot, and only detects the end of the vacuum gripper. Note two facts. The first one is that the RGB does not need to have this threshold on depth. The second is that this snapshot is illustrative, it does not correspond to a snapshot that will be used to process the points of pick.

As it has been shown in 5.1 the cameras work like a radial camera. This drives two difficulties that may drive problems. The first one is that the position of a pixel in a photo RGB does not directly determine the position of a point. Additionally, the depth is not directly given by the value of an image of depth. Illustratively, this can be observed in 2D in Figure 31:

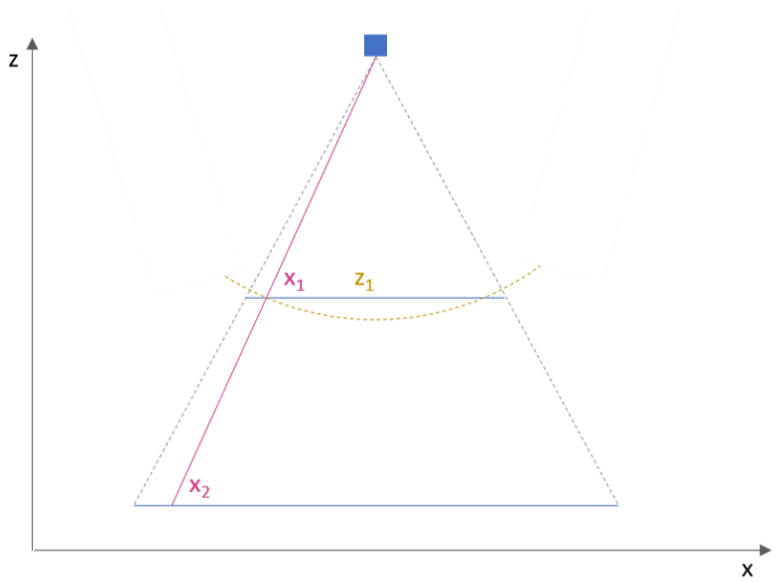

Figure 31 Representation of the biases produced by the radiality of the camera

In the mentioned figure, the blue lines represent two different surfaces that the robot may detect, for instance the floor and the top of a cylinder. The grey lines represent the limits of the camera, separated a given angle. The pixel taken in a specific moment can be represented by the purple line. Firstly, for the same angle opening, the position of the point x_1 and x_2 is different. Secondly, the value of height z_1 in orange represents the height of the upper surface, which will not be the value the camera gives, as for the camera the height (or depth in this case) represents the distance between the point and the camera.

Using trigonometry, and neglecting the error in depth of the camera, it can be shown that:

$$z = z_{min} + \frac{z_{max} - z_{min}}{P_{z-max} - P_{z-min}} (P_z - P_{z-min}) \tag{27}$$

$$R = R_O + 2\,(H - z) * tg\left(\frac{\alpha}{2}\right) * \left(\frac{P_x}{(P_{x-max} - P_{x-\min})} - 1/2\right) \tag{28}$$

5.2 ALGORITHM ANALYSIS

In this chapter the operation of the algorithm will be explained step by step, firstly describing the overview of interaction with the environment and secondly zooming in the reinforcement learning algorithm.

The overview of the whole algorithm is described in the following pseudo-code:

1. Do a snapshot of the current pick box in RGB and depth
2. Find the contour of the box using the depth photo
3. Create a grid of points equidistant and separated equally from the sides of the box
4. For every point:
    a. Tag the point as eligible if its depth is below a certain threshold
5. If there is no eligible point, tag the other box as the pick box and the pick box as place box
6. Compute epsilon using a uniformed distribution from 0 to 1
7. Update alpha using the epsilon greedy strategy (starting with 0 and ascending exponentially)
8. If epsilon is greater than alfa, tag the action as exploration, exploitation otherwise
9. Select the action:
    a. If exploration: Point selected randomly
    b. If exploitation: Compute the probability using the Neural Network for all the eligible points, and select the one with the highest probability
10. Order the robot to go to that point
11. Order the robot to try picking the object and save whether it was picked or not
12. Finish the robot movement:
    a. If the object was picked: Place it in the place box and return to the security position
    b. If the object was not picked: Return to the security position
13. Use the CNN to get the information about the snapshot
14. Update the Replay Memory adding the episode just experienced
15. Run the Neural Network:
    a. If number of experiences is lower than 10: The NN does not train
    b. Else if number of experiences is lower than the batch size: Train the NN with number of experiences as batch size
    c. Else run the NN with batch size
16. Save rewards of the episode and accuracy and loss of the NN training

5.2.1 OVERVIEW OF THE ENVIRONMENT MANAGEMENT

At the start of each episode, a snapshot of the box of pick is taken, and is sent to the agent through a publisher in Lua and a subscriber in the python file. The figure that is sent is the raw image of Figure 32:

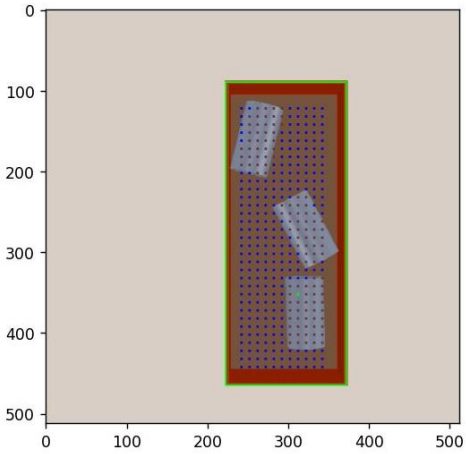

Figure 32 Preprocessed photo of the box with contour, grid and the point to pick

This figure is a preprocessed image of the snapshot taken. Firstly, the algorithm has painted a green rectangle to determine the contour of the box. Secondly, the algorithm creates a grid of points separated from sides of the box with a similar padding. Finally, it evaluates the value of depth of those points and eliminates those points with a depth higher than a threshold (blue points). Therefore, the red points of the image are those who are eligible to be picked by the robot. If there are no red points in the grid the algorithm understands that the box is empty and thus tags the empty box to full box and vice versa (or place box to pick box).

Once the agent has this information, it selects a point out of all the red points, by exploration or exploitation, depending on the epsilon greedy strategy.

Then, by using again the publisher-subscriber method, the agent publishes all the actions that the robot should do to arrive at the point selected. The Lua code receives the orders by the subscribe method and executes them. There are two different pub-subs, one for the direct kinematics and the second one for the inverse kinematics.

At a specific moment, when the robot should have picked the object and has lifted, the robot informs the agent whether the object has been picked or not, and the information about the image is saved, tagged with fail or success depending on the outcome.

Finally, the robot places the object in the other box and return to the safety position.

5.2.2 OVERVIEW OF THE REINFORCEMENT LEARNING ALGORITHM

In this chapter the modules related with the reinforcement learning algorithm will be explained.

5.2.2.1 Epsilon Strategy

The epsilon greedy strategy is the function that determines whether the episode will be a episode of exploration or exploitation. In the first episodes of the training, the agent should predominantly choose exploration, as there is little to no information about the environment.

Afterwards, as the Neural Network is more trained and the number of experiences in the replay memory grow, the agent should choose more and more exploitation.

There are two parameters that determine this decision:

- Alpha: For every episode alpha is calculated using a uniform distribution between 0 and 1
- Epsilon: Epsilon starts being 0 and tends exponentially to 1

The equation for alpha is the following:

$$\alpha \sim U(0,1) \tag{29}$$

And the equation for epsilon is the following:

$$\epsilon = 1 - e^{-\beta i} \tag{30}$$

With β being the actualization rate and i the episode of the training. Nevertheless, as the parameter β does not clearly represent the behavior of the epsilon greedy strategy, the percentage decay is presented, such as:

$$p = -\frac{\ln(0,5)}{\beta N} \tag{31}$$

Being N the number of episodes that will be done in the training. Thus, the percentage decay represents at which percentage of episodes epsilon is 0,5, having the same probability of doing exploration and exploitation.

Figure 33 represents the evolution of epsilon across the episodes in a training, for a percentage decay of 10%:

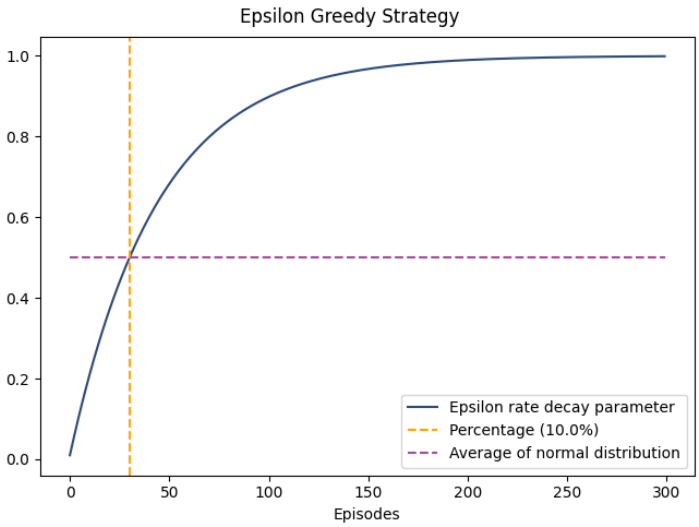

Figure 33 Epsilon greedy Strategy graphical representation

As it can be seen in the image, the average of alpha, which follows the uniform distribution, is 0,5. Therefore, when epsilon is higher than 0,5 it is more probable that the episode will be exploiting the Neural Network.

If this project entered in a production phase, some adjustments of the epsilon greedy strategy could be put in place to increase the algorithm robustness to the variability of the environment.

If this algorithm was deployed in a warehouse, the environment could change by a wide variety of reasons (new objects, new color of the box, etc). In that case, the robot could continually exploit the environment, but not have the correct optimization weights in the NN.

As a countermeasure of this hypothesis, if the epsilon was above 0,5 and the robot average reward over the last 20 episodes was below 50 %, the epsilon greedy strategy could be automatically reset and the robot would start exploring the "new" environment.

5.2.2.2 Replay memory

Replay Memory is the python object that stores the experiences that will be used by the policy (Neural Network) for training. It is initialized empty as there has not been any experiences covered before. As it has been mentioned, an experience is formed by a tuple:

- Input: Contains the information of the image, in the format of a vector of 1000 real numbers. It is the output of the Resnet.
- Output: Is the reward obtained when going to that point in the past.

Every time the algorithm completes an episode, it stores the experience in the replay memory, as it will be used in the Neural Network Training.

The replay memory has two hyperparameters:

- Batch Size: Is the optimal number of experiences selected to train the Neural Network. A high batch size may be interesting where computer resources is not a constraint, as the higher the batch size the more the Neural Network is trained. A low batch size may drive low convergences.
- Capacity: The number of experiences that will be stored in the object. A high capacity can be suitable to always select random samples, as the higher the rate capacity/batch size, the lower probability has a sample to be picked. Nevertheless, a lower capacity may be suitable to changing environments, where the first samples of the training are outdated, and for eliminating biases, as it is likely that the first images will have been trained more than the last images (they have had the chance of being picked more episodes).

In the base case, when the Neural Network is going to be trained, the experiences are selected randomly. Nevertheless, this project will analyze to other possibilities to increase the overall performance of the Neural Network:

- Indecisive cases: If the Neural Network is computing a probability of around 0,5 for a point in the replay memory, it means that it does not know whether it is a success or a failure. Thus, to force the robot to eliminate these indecisive cases, the experiences whose probability are closer to 0,5 are selected as the training dataset.
- Worst cases: If the Neural Network is computing a probability next to 0 for an experience that is a success (1), or is computing a probability next to 1 for a failure (0), it means that the NN is convinced of something and is wrong about it. Thus, to force the robot to correct the cases in which it is wrong, for every point, the absolute difference between the probability computed and its reward is calculated those with the highest value are selected to be trained in the NN.

5.2.3 CNN AND NEURAL NETWORK

As it has been mentioned before, the CNN Resnet takes as input a RGB snapshot 50x50, in the form of a tensor (3x50x50) and outputs a vector of length 1000, containing the information about the image.

Then the Neural Network takes this vector as the input layer. The NN have two hidden layers, of 256 and 32 nodes, and one node in the output layer, to compute the probability of success or failure.

The Neural Network uses one parameter, which is the learning rate. As it has been explained in 2.2.1, a high learning rate will trust more the new trainings. Therefore, a high learning rate can drive a faster convergence, but it can lead to higher variance in the final results.

5.2.4 REINFORCEMENT LEARNING SENSITIVITY

All the hyperparameters described above will be optimized by conducting a sensitivity analysis. To do so, for each hyperparameter, the algorithm has been executed using a grid of the possible values of the hyperparameter, and the results have been analyzed to find the optimal hyperparameters. Some considerations:

- The value of the hyperparameters before optimization are the base case values
- The sensitivity analysis has been conducted for one parameter at a time, which may not be the optimal procedure, but is the fastest process to improve accuracy
- In order to reduce variability, the results have been run twice for every value of every hyperparameter

Chapter 6. ANALYSIS OF THE RESULTS

6.1 BASE CASE RESULTS

The base case results use the original hyperparameters, that were set before analyzing the performance of the algorithm. They are set based on prior the optimal parameters for other similar models. Therefore, even though they are not optimal, they should be directionally aligned with the final model.

For this algorithm, the base case hyperparameter's values are the following:

- Percentage Decay: 30%
- Learning Rate = 0,001
- Batch Size = 80
- Capacity = 10.000

The base case results can be found in Figure 34:

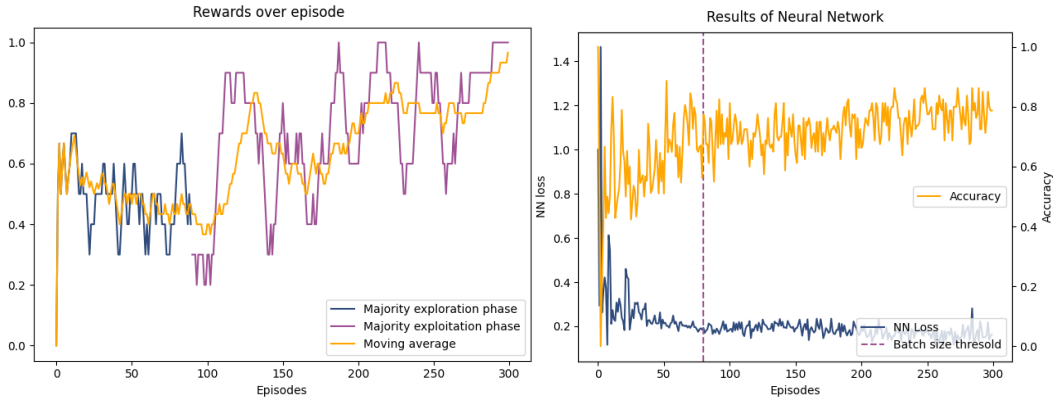

Figure 34 Base case results

The figure in the left represents the average rewards over the episodes, for a moving average of 10 (blue and purple lines) and for a moving average of 30 (yellow line). For the first n episodes, where $n < T_{mov.avg}$, the moving average is of period n. Rewards cannot be provided without a moving average as the plot bounces between 0 and 1. Note that the higher the period of the moving average, the higher the delay of the plot. Additionally, note that the blue line is built mostly by exploration episodes whereas in the purple exploitation predominates.

The figure on the right represents the results on the Neural Network. The blue line shows the resultant loss value of the optimization in each episode, whereas the yellow line shows the accuracy of the selected batch of experiences after optimizing the algorithm. (Percentage of cases predicted to 1 when reward is 1 and predicted to 0 when reward is 0). Finally, the purple line plots the percentage decay, meaning the moment where the rewards are mostly due to exploitation.

On the left graph there are three remarkable comments. The first one is that the rewards are generally growing across the episodes, which means convergence of the algorithm. The second comment is that even with a moving average of 30, the plot does not grow steadily, which means that there is a variability in the results. The third comment is that the final rewards are close to 1, which would mean that the algorithm is predicting correctly 10 cases in a row. Nevertheless, this could be just a coincidence given the variability already mentioned.

In the graph on the right it seems the loss function and the accuracy are not growing after passing the percentage decay, which could mean that the maximal accuracy in this model is 80%, and that there is no further improvement of results after the 80^{th} episode.

6.2 SENSITIVITY ANALYSIS

6.2.1 PRESENTATION OF THE PLOTS

For every sensitivity analysis three different plots will be shown. Before analyzing every sensitivity analysis, those plots will be explained without emphasizing in the results of the plot, for a better understanding. The first example is Figure 35.

The first plot corresponds to the average of the last 10 rewards for every training. Therefore, this plot represents how the algorithm is performing in the last episodes. The second plot shows the first episode for every training in which the accuracy in the training batch is higher than 0,9. Hence, this plot shows how fast the algorithm is converging. The third plot shows the last values of the accuracy and loss of the Neural Network, representing how well the Neural Network has been optimized through the episodes.

6.2.2 PERCENTAGE DECAY

The first hyperparameter that will be optimized is the percentage decay. The percentage decay shows at which percentage over all the episodes the algorithm there is a higher probability of having exploitation rather than exploration.

The base case of the percentage decay is 30%. The first grid of values of percentage decay are in the range 5%-50%:

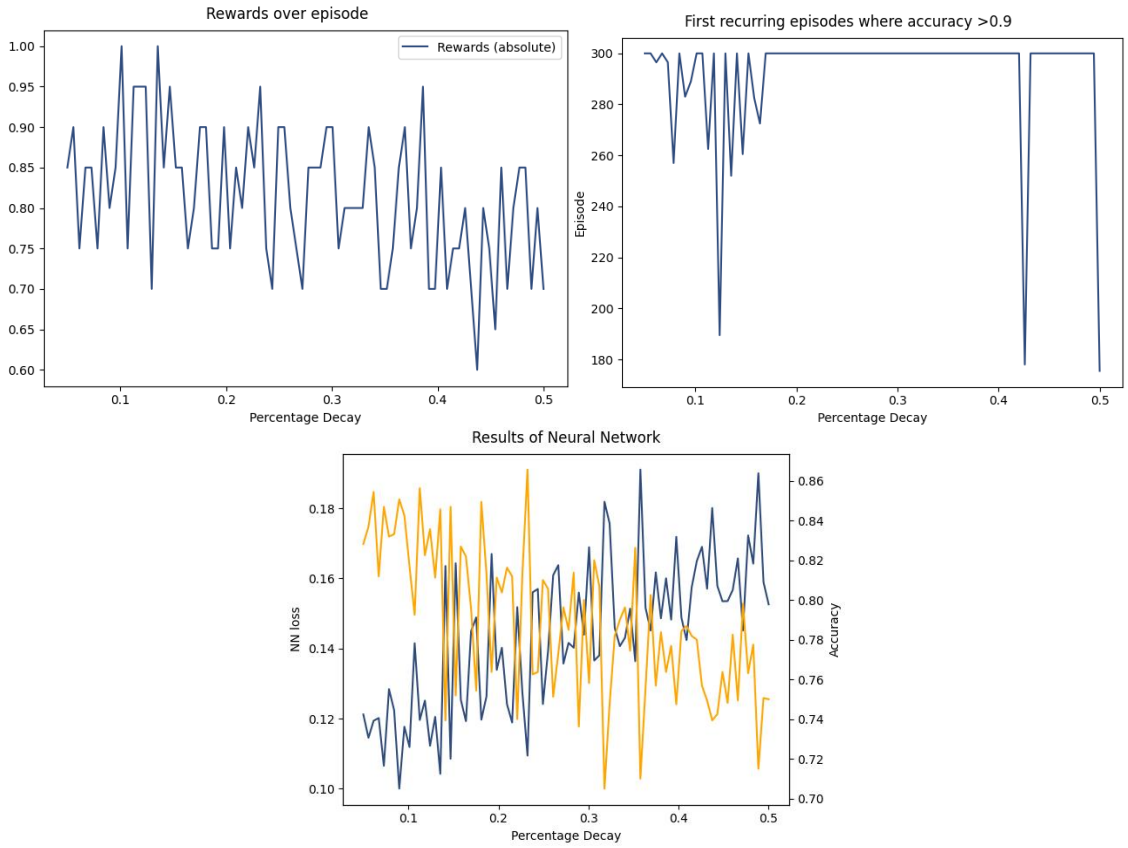

Figure 35 Results of the first sensitivity analysis of the percentage decay

Analyzing the second plot, it is clear that a lower percentage decay will probably give better results, as the convergence is faster. Additionally, analyzing the third plot it is clear that a lower percentage decay gives higher terminal values. Thus, another sensitivity analysis is performed in Figure 36for a range of 5%-20%:

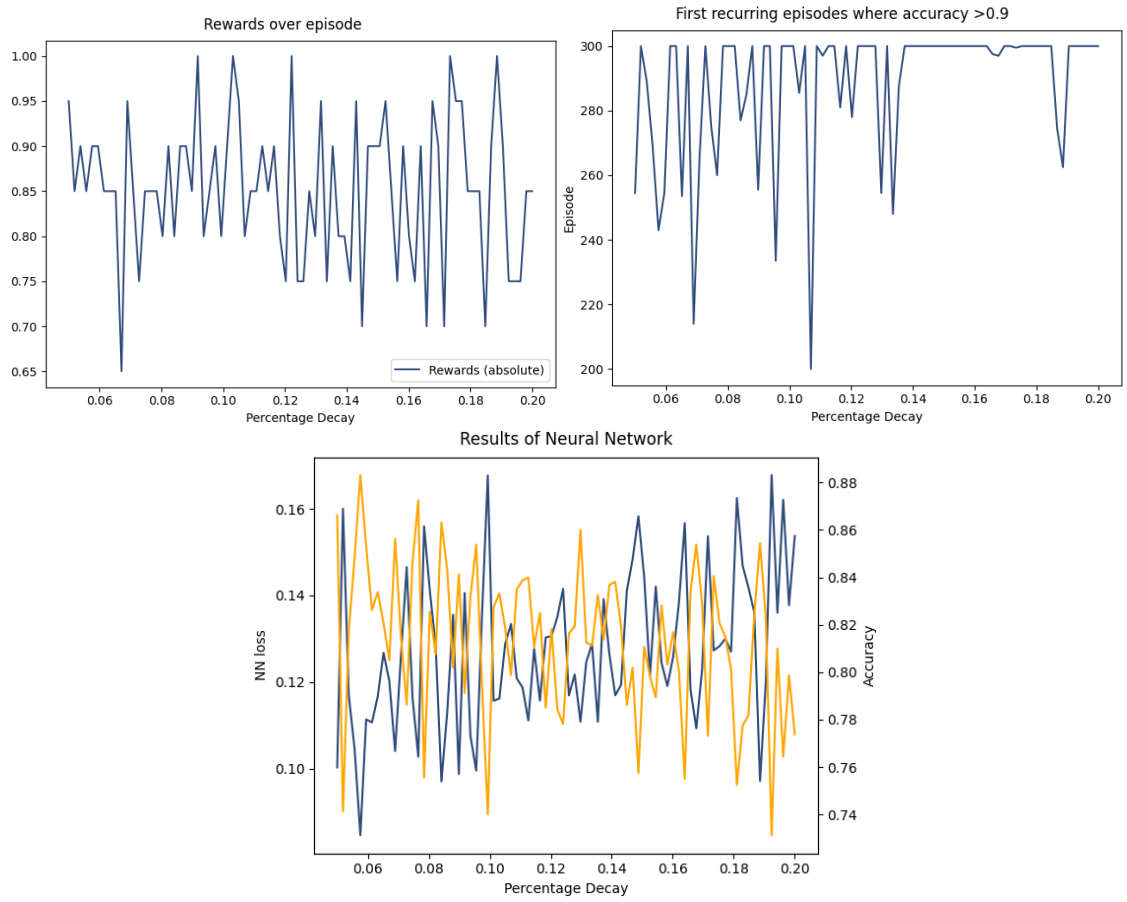

Figure 36Results of the second sensitivity analysis of the percentage decay

Even though results are not clear, looking at the second plot, it seems that the optimal parameter is between 5% and 10%. Therefore, the parameter selected is 8%. The new results are those of Figure 37:

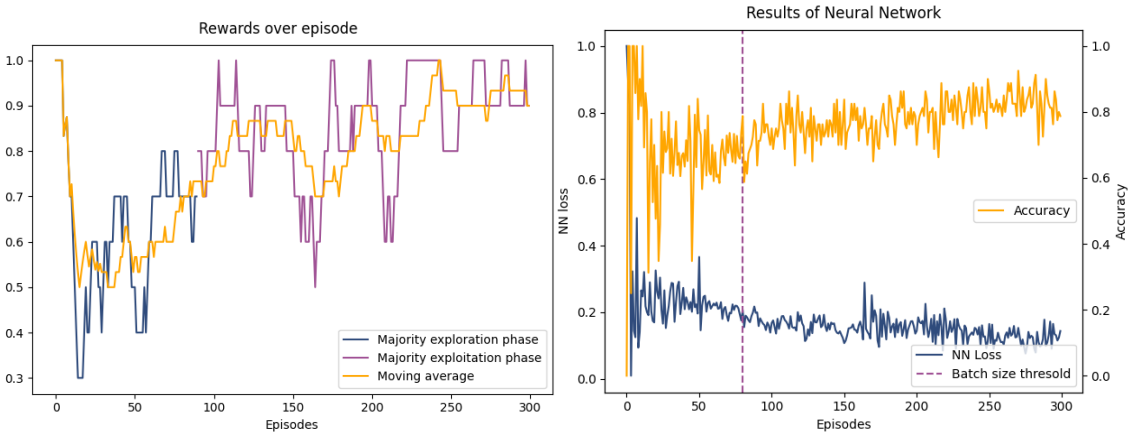

Figure 37 Results after percentage decay optimization

Even though the graph on the left is similar to the base case plot, in the graph on the right the Neural Network seems to be learning slowly but steadily after the episode 100. Figure 38 compares the base case results with the current result:

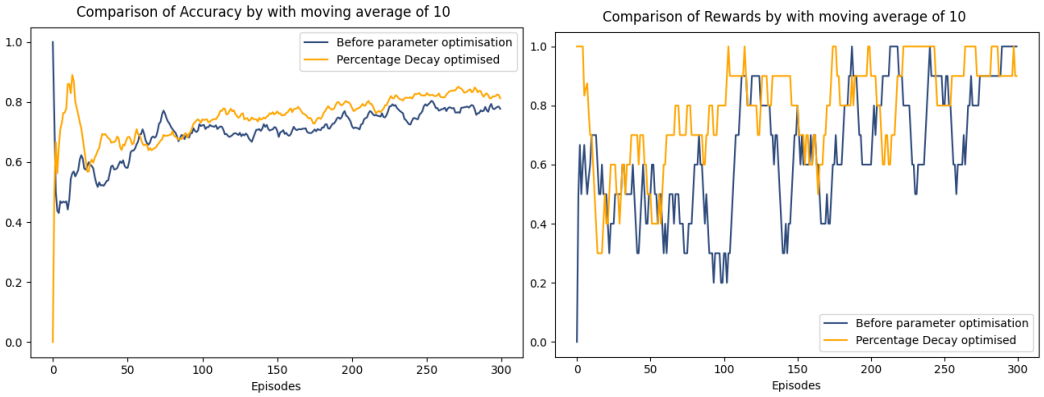

Figure 38 Comparison between results before and after percentage decay optimization

The plot is now having a faster and higher accuracy in every episode, unless around the 80th episode (may be due to variability), and therefore this hyperparameter is better than the previous one.

6.2.3 LEARNING RATE

As it has been mentioned, a higher learning rate may result in better algorithm performances, but with the counterpart of a high variability. The base case result is 0,001, and the grid performed is between 0,00003 and 0,02:

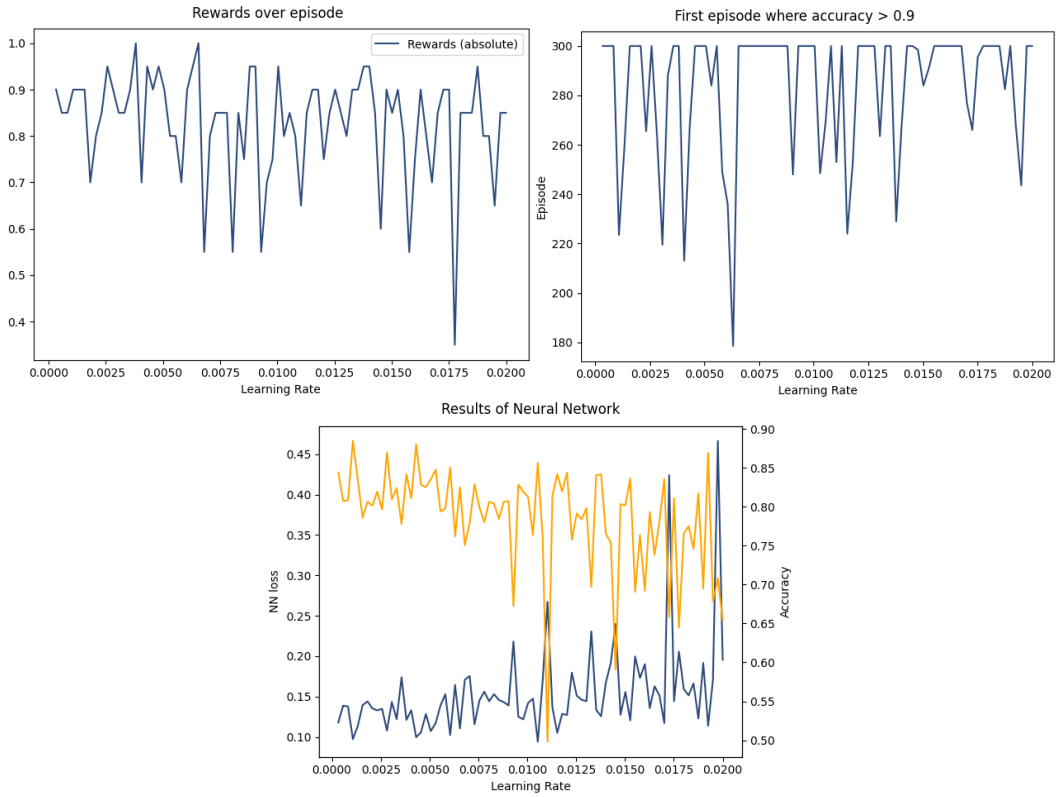

Figure 39 Results of the sensitivity analysis of the learning rate

A lower percentage decay seems to have faster convergences (second plot) and a better terminal value (first plot and third plot). Nevertheless, this could be due to variability and the base case hyperparameter, 0,001 seems to be already a good choice. Thus, the hyperparameter will not be changed.

6.2.4 MEMORY CAPACITY:

Memory Capacity handles the amount of experiences that can be stored in the Memory Replay at the same time. A low memory capacity promotes image rotation in the training batch. Nevertheless, a high memory capacity allows having more images to pick in the training batch (more random batches, or better decision of indecisive/worst case experiences)

The current algorithm has a memory capacity of 10.000 (which is similar to infinite, as the number of episodes is 300) and the grid performed is 120-300:

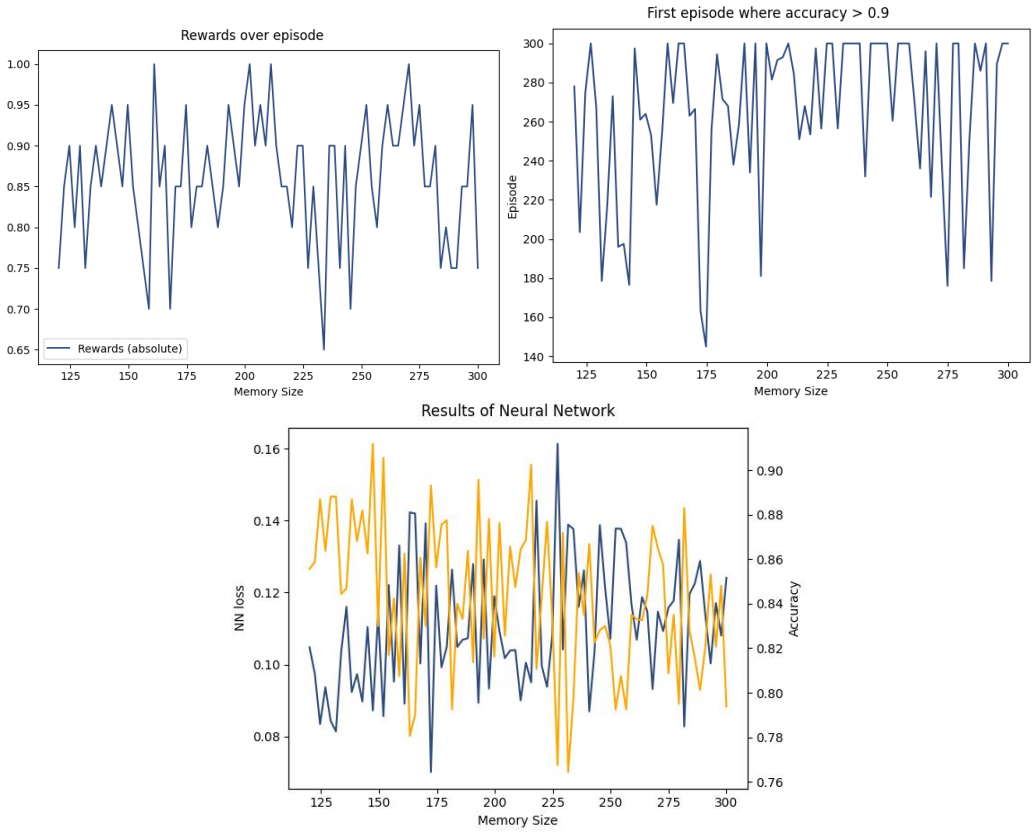

Figure 40 Results of the sensitivity analysis of the memory capacity

Even though the results are not clear and capacity memory should not greatly affect the overall accuracy of the algorithm, capacity memory of 150 has been chosen to compare the accuracy between both results. The plots of the new algorithm are those of Figure 41:

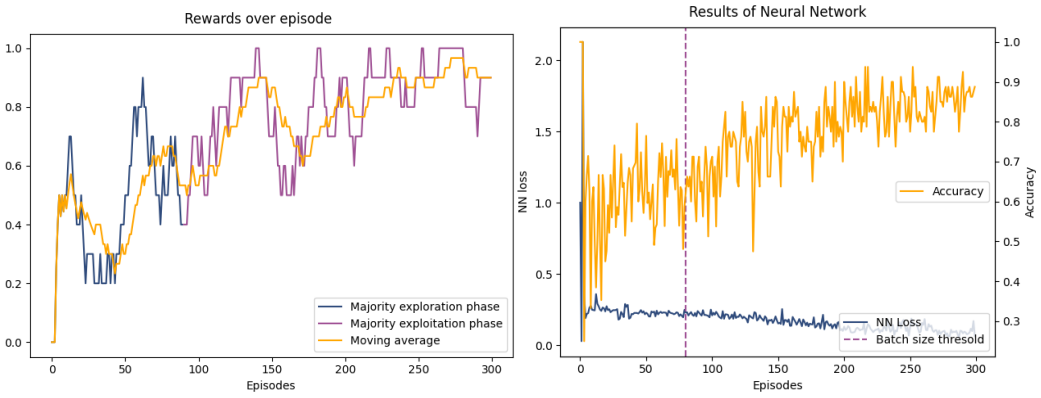

Figure 41 Results after the capacity memory optimization

The image on the left is not showing clear results, even though it seems that the accuracy is higher than 95% for the last episodes. In the right plot, the slope of growth across episodes is also positive, which means that the algorithm is improving.

The comparison of this result with the previous configuration of hyperparameters is shown in the following figure:

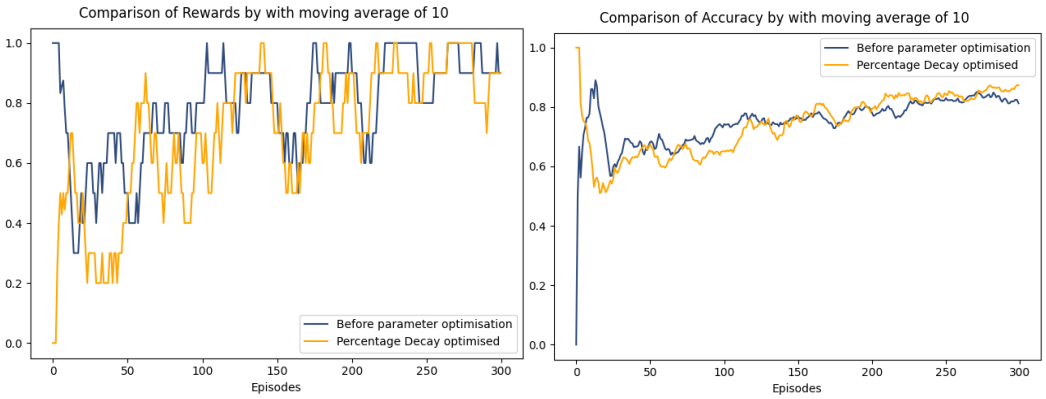

Figure 42 Comparison of results before and after capacity optimization

As predicted, the convergence is similar in both cases.

6.2.5 BATCH SIZE

The batch size is the hyperparameter that indicates the number of experiences selected for training at every episode. A high batch size may drive a faster convergence, but a require higher computation resources. A sensibility analysis has been performed in Figure 43 for a grid of 40 to 160, to analyze to what extent the results improve as the batch size grows:

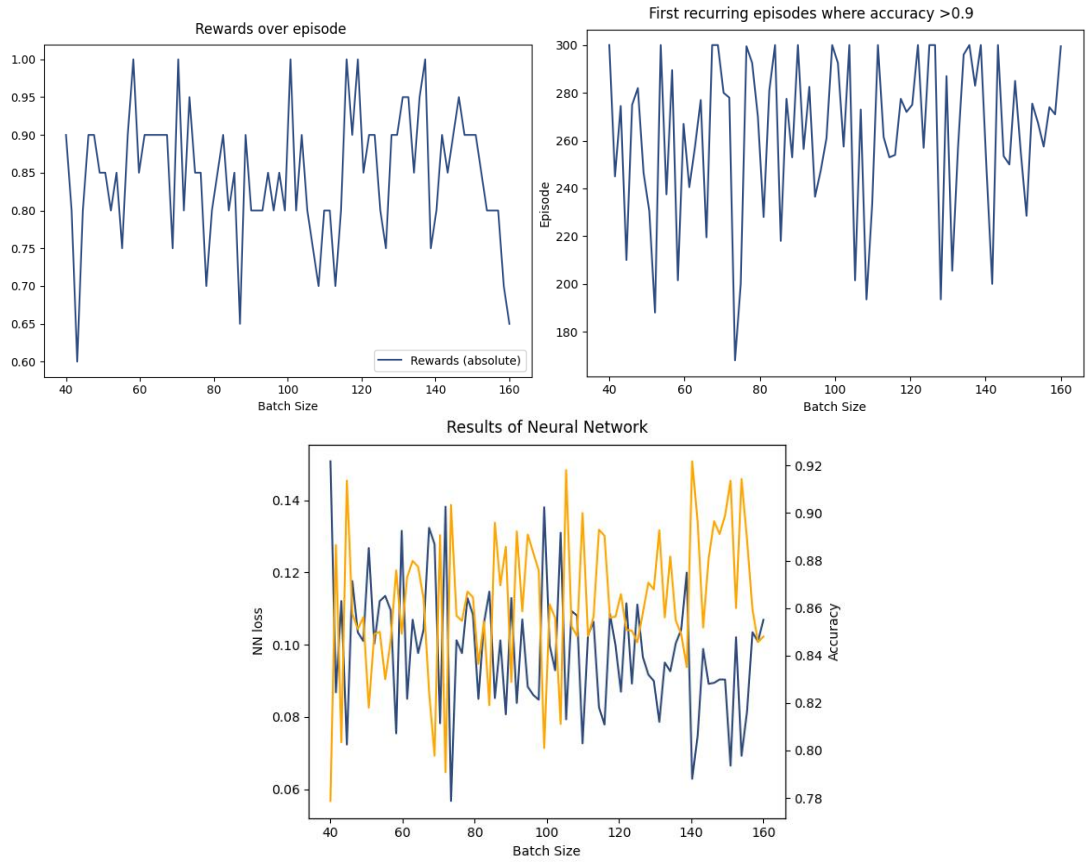

Figure 43 Results of the sensitivity analysis of the memory batch size

The results do not seem to prove that the performance is better with a higher batch size, neither with a lower. As a consequence, no change of the batch size has been done and the previous result is the final result.

6.3 REPLAY MEMORY STRATEGIES:

6.3.1 PRESENTATION OF EACH STRATEGY

As it has been explained in 5.2.2.2, there current algorithm could be modified to perform better in the current scenario. In the formal presentation of a reinforcement learning algorithm, the training batch is selected randomly. Nevertheless, a smart selection of the experiences to be trained could produce a better result. Concretely, the smart selection could be one of this two possibilities:

1. Worst case Sampling: Name given to the selection of the experiences with the highest error between the probability computed by the NN and the reward
2. Indecisive sampling: Name given to the selection of the experience whose probabilities computed by the NN are closest to 0,5.

6.3.1.1 Random Sampling:

Figure 44 shows the results of the algorithm for the random sampling, which is the sampling used for optimization.

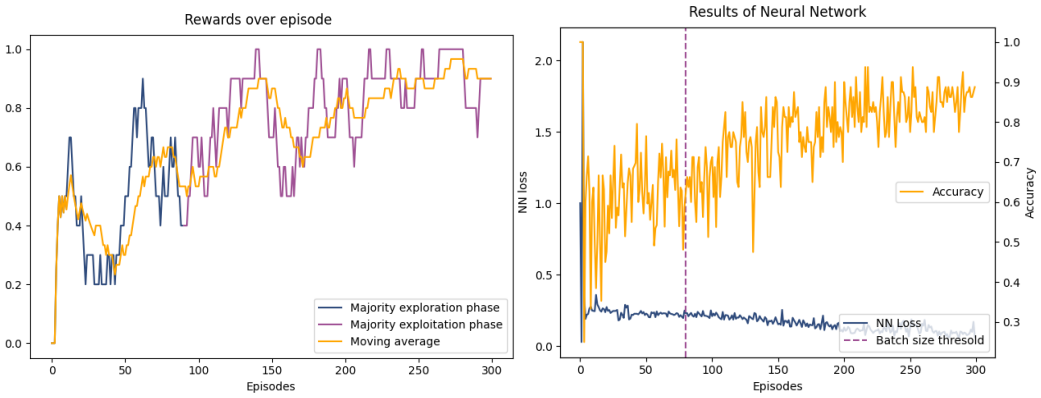

Figure 44 Results of random sampling after optimizing hyperparameters

As commented, the overall reward is bouncing but overall growing and the accuracy and the loss are improving gradually and steadily.

6.3.1.2 Worse Case Sampling:

Figure 45 shows the results of the worst-case sampling, for the optimal hyperparameters in the random sampling:

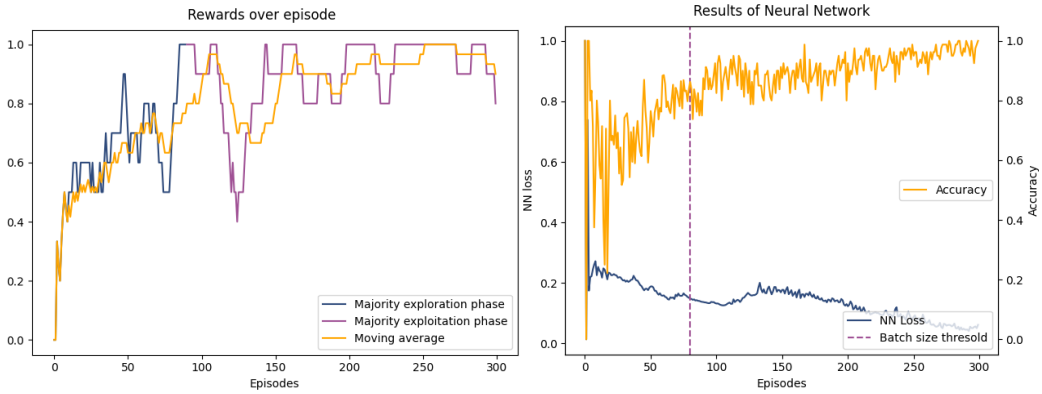

Figure 45 Results of worst-case sampling using optimized hyperparameters

There are several comments to do about this figure. First of all, looking at the picture on the left, the rewards seem to be high in the first episodes and, even it bounces, it seems to be converging at a reward of around 95 %. Secondly, the loss function overview quite different from those seen until now. The loss descends heavily in the first 100 of episodes, then it is steady (or even the loss grows) from the episode 100 until the 150 and finally it descends

again with a high slope. Additionally, the scale of the loss plot is from 0 to 1, whereas in the random sampling is from 0 to 2, so the loss is much lower in this case. (Reason?)

Regarding the accuracy, it is quite clear that the algorithm converges fast into an accuracy greater than 80% (around episode 100) and then grows steadily until almost 100%.

6.3.1.3 Indecisive case Sampling:

These are the results of the indecisive case sampling, for the optimal hyperparameters in the random sampling:

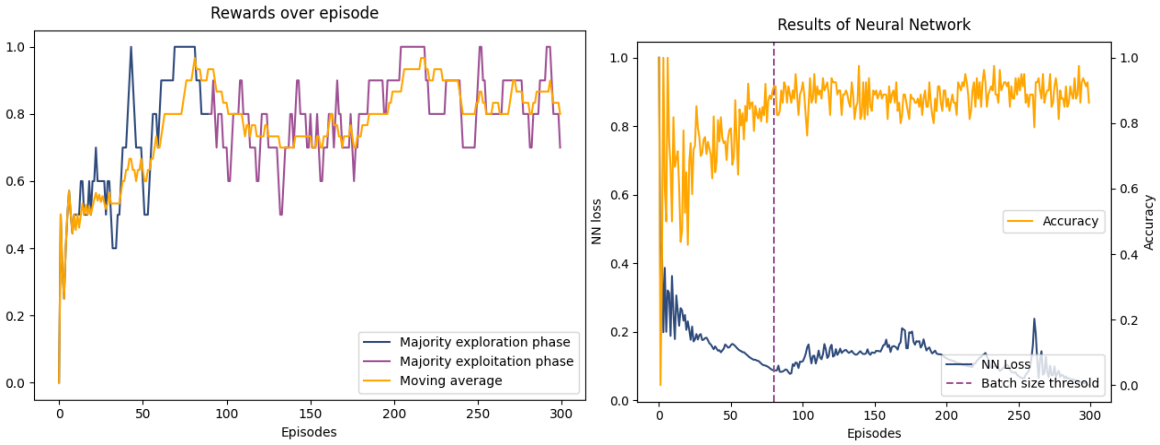

Figure 46Results of indecisive sampling using optimized hyperparameters

In the left plot, the reward results seem to have a lower variance and stay around 85%. On the plot on the right side, the loss plot is showing a different shape than what it has been seen so far.

The convergence in the first 100 episodes seems to be the highest. Then, for the 80 consecutive episodes, the loss grows, and then descends again with some peaks of loss around the episode 260. The reason of this behavior could be the fact that the training batch has no more indecisive photos to take, as they are all above 0,7 or below 0,3. Thus the idea is no longer working and the dataset could be unbalanced (more photos of one category fail/ success than the other).

6.3.2 COMPARISON OF THE STRATEGIES

In this chapter the plots presented above will be compared to decide which is the best possible scenario. The following plots compare the worst-case sampling with the random sampling (which is the base case in this case).

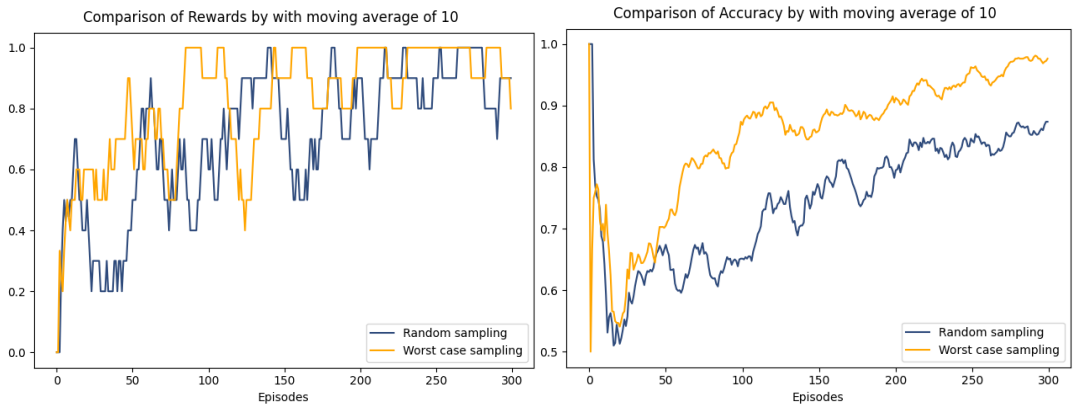

Figure 47 Comparison between random sampling and worst-case sampling

Even though it is recognizable in the left plot that the worst-case sampling is better, the comparison is clear on the plot on the right. It has a faster convergence and finishes the episode 300 with an accuracy next to 95%, whereas the random sampling is almost 90%.

Figure 48 compare the random sampling with the indecisive sampling:

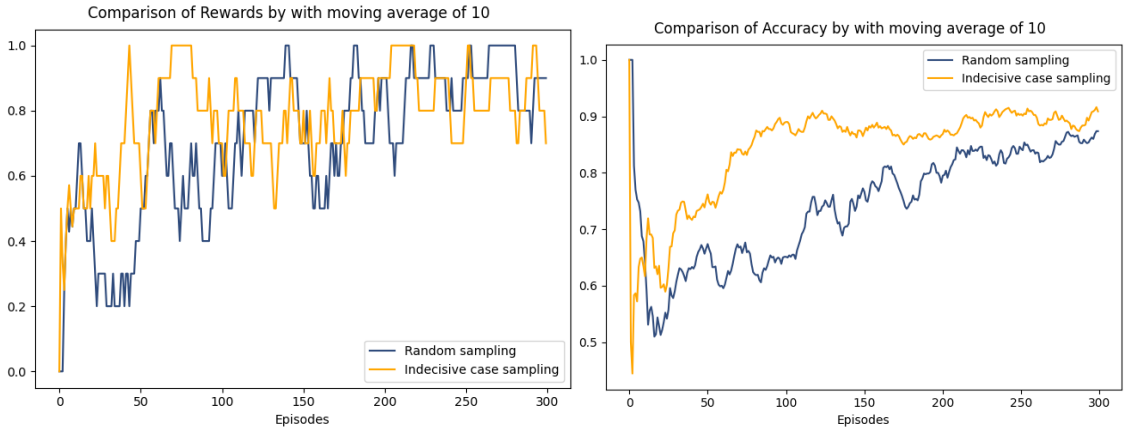

Figure 48 Comparison between random sampling and indecisive sampling

Again, the reward plot on the left shows a faster convergence, even though the left plot is showing the results far clearer. The convergence is faster on the indecisive sampling and the final accuracy is close to 95%, higher than the random sampling.

In the following figures the three cases are compared:

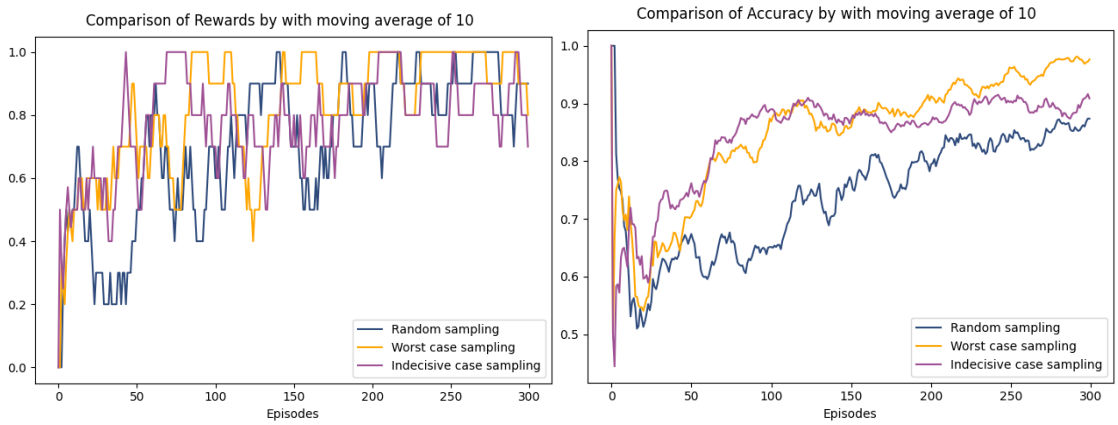

Figure 49 Comparison between random, indecisive and worst-case sampling

The right figure shows that the random sampling is not best in any case. When comparing the indecisive results with the worst-case sampling, the indecisive seems to have a faster convergence, whereas the final accuracy of the worst-case sampling is much higher.

As a consequence, it may be interesting to have a model which starts optimizing the NN with the indecisive sampling, then with the random sampling and finally with the worst-case scenario. This will be named onwards as dynamic sampling.

Dynamic sampling has been calculated for a 30% /40% /30% split indecisive, random and worst case (in that order) and 15% /35% /50% split (as the worst case is the one giving the higher accuracies). Figure 50 shows the results for the first case:

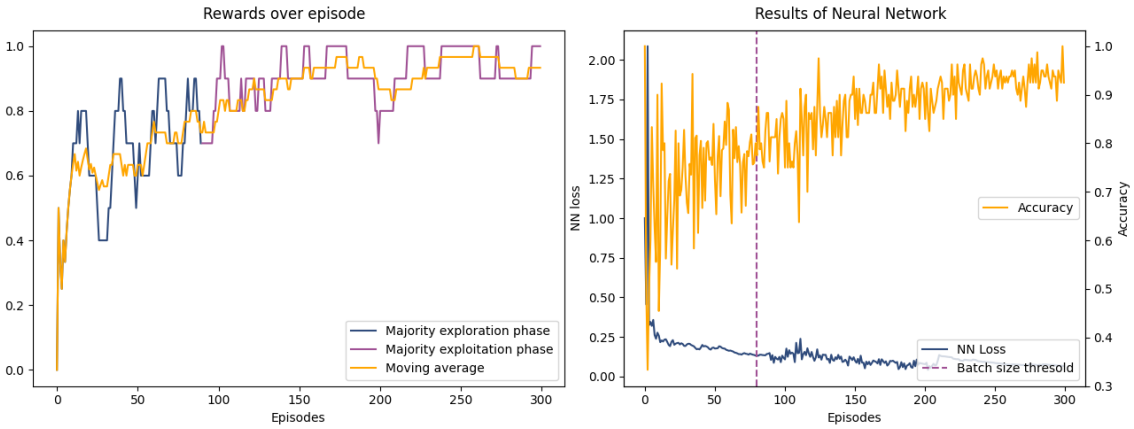

Figure 50 Dynamic sampling for 30%/ 40%/ 30% of indecisive/ random/ worst-case sampling

The results seem promising regarding the plot on the left, as the convergence is fast and the final accuracy arrives in episode 150 with 95%. The loss function seems to converge steadily in the indecisive phase, but in contrast to it, does not overshoot between episodes 100 and 180. It slightly overshoots around episode 220, probably because the worst-case sampling

has started, but keeps descending right after. The accuracy, even though is growing and finishes around 95%, is quite noisy.

The figures for the dynamic sampling for the second split are in Figure 51:

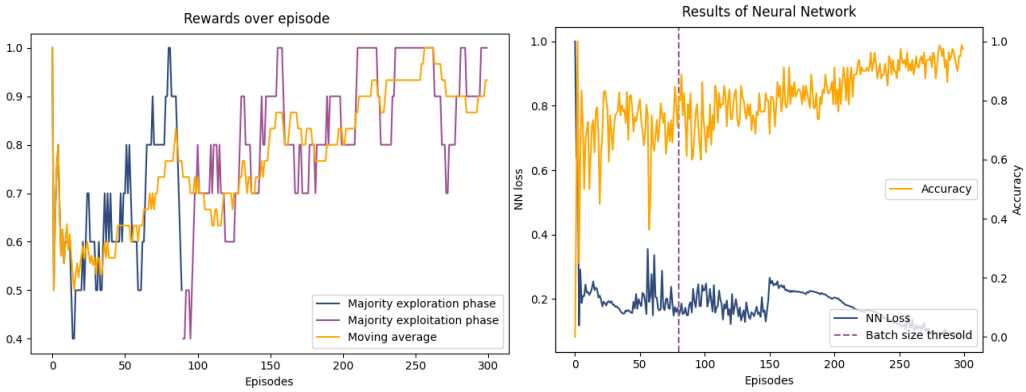

Figure 51 Dynamic sampling for 15%/ 35%/ 50% of indecisive/ random/ worst-case sampling

The results seem to be worse than in the previous case. The indecisive case probably has not has the enough episodes to learn, ant there is an overshoot when turning to random sampling (it happens in episode 45). Likewise, in episode 150 the worst-case sampling starts and there is an overshoot in accuracy.

For the accuracy, it seems that only advantage of this model is the final accuracy, next to 95%, probably only due to the worst-case sampling.

Figure 52 show the comparison of both models:

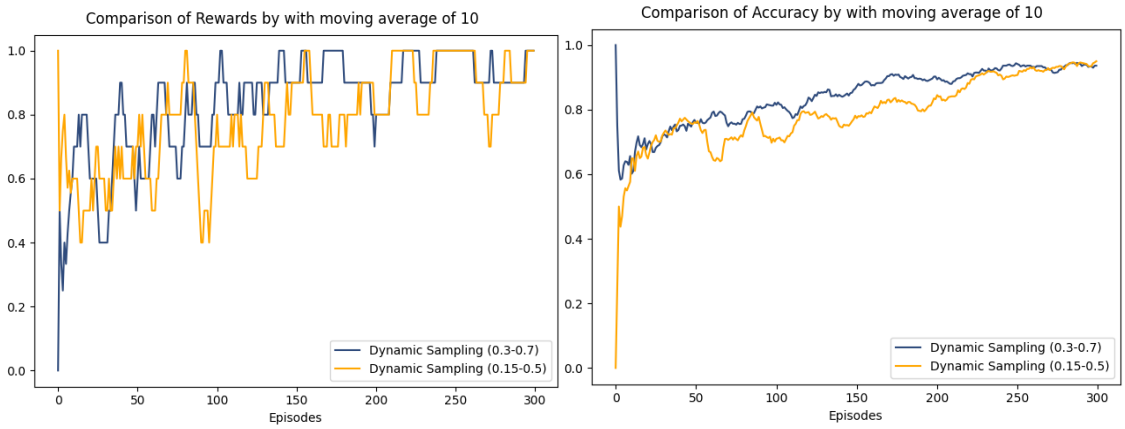

Figure 52 Comparison between the first dynamic sampling (0.3/0.7) and the second (0.15/0.5)

By looking at Figure 52 on the right, the first option of dynamic sampling seems to be better as it is converging faster. Nevertheless, both models have the same final accuracy which could be lower than the worst-case scenario.

Finally, all the models commented above have been compared between each other in Figure 53:

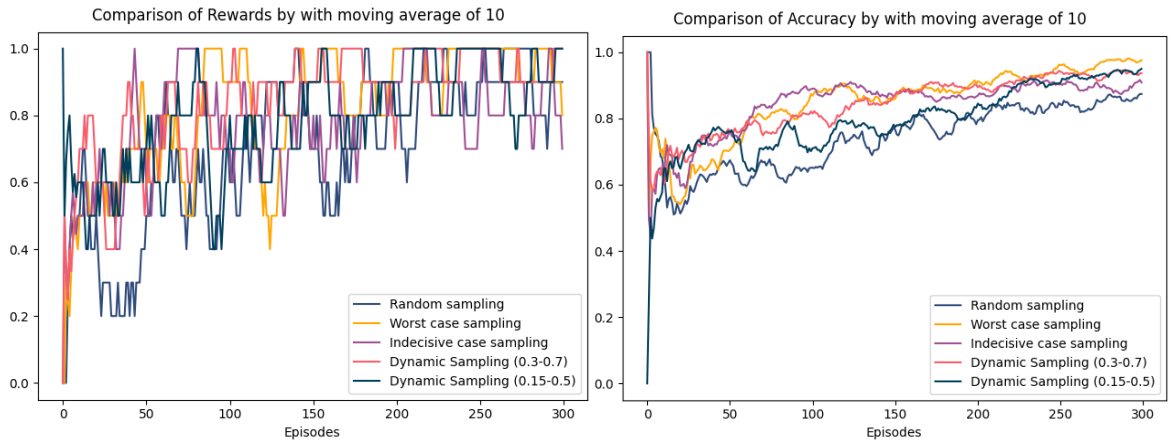

Figure 53 Comparison of all the Replay Memory Strategies

The figure on the left is shown to maintain the display of all the plots in the report, even though does not show anything given that the results are too superposed. Nevertheless, in the image on the right, the model which seems to be converging faster is the indecisive sampling, with a convergence of around 90% in the episode 100, but the worst-case sampling has the best accuracy of everyone. Therefore, if this model was to be implanted in a warehouse, the worst-case sampling would be selected.

Chapter 7. CONCLUSION AND NEXT STEPS

The purpose of this chapter is to review the completion of the objectives of the project and discuss about new ideas which could be implemented to improve the performance of the algorithm, performing in a physical environment, eventually a warehouse.

7.1.1 CONCLUSIONS OF THE PROJECT

Following the objectives presented in 4.2, the conclusions are the following:

7.1.1.1 Virtual Environment

Exploring and testing the software, as well as investigating the sources of the software Sim Coppelia have allowed to deploy the main functionalities into this project and create a digital twin of the laboratory.

The robot behavior (joints, link, direct kinematics, inverse kinematics, etc) mimic perfectly the normal operation of an anthropomorphic robot, such as the one in the laboratory. Additionally, the robot used is the exact representation (number of joints, type of joints, length of the links, etc) of the robot in the laboratory, which is the UR5.

In terms of the objects used, the shape and the orientation of the object is similar to the objects used in the laboratory. The dynamic behavior of the objects is also an acceptable representation of the physical world.

Nevertheless, there are two features of the robot that could be improved to better represent the reality: the vacuum gripper and the sensor that detects if the object is attached. The vacuum gripper only grips an object if the point if the middle of the tool is touched by the object. In that case, the base of the gripper may not be entirely covered by an object, and still pick the object (something impossible in a real vacuum gripper). Additionally, the sensor that detects if an object is gripped corresponds to the same point, in the middle of the vacuum gripper. The object is detected only if the object is touching the middle of the base of the vacuum gripper, which could not always be the case in the real world (Due to pression in the vacuum gripper the height of the base could change).

7.1.1.2 Learning Algorithm

In terms of the learning algorithm, the reinforcement learning algorithm has been correctly deployed and gives promising results in the shown environment. After performing a sensitivity analysis of the hyperparameters, the reinforcement learning applied for the base case, with random sampling, provides an accuracy between 80% and 85% in the episode 300 (and is still growing).

The hyperparameters for the base case where the following:

$$(p, lr, N, n) = (30\%, 0.001, 10000, 80) \quad (32)$$

And for the optimized case, the following:

$$(p, lr, N, n) = (8\%, 0.0001, 150, 80) \quad (33)$$

Corresponding to the percentage decay, the learning rate, the memory capacity and the batch size of the Replay Memory.

Additionally, when testing new strategies of sampling the experiences that are going to be used for training, the worst-case sampling method provides a growing accuracy between 95% and 100% in the episode 300, and a steady accuracy of 90% for the indecisive case.

7.1.2 NEXT STEPS

Over the whole report some improvements of the current state of the project have been mentioned. These will be grouped and explain thoroughly in this chapter:

7.1.2.1 Vacuum gripper

Even though not deployed in the current state of the project, it seems that there is a possibility to create a physical vacuum gripper, similar to the one in the laboratory. If the environment had a more realistic vacuum gripper, the project would improve in two aspects. The first one, is that, as the vacuum gripper would behave as the physical tool, the success/fail trials would be similar to the successes and the fails in the physical environment, and thus the dataset to train the Neural Network would be more realistic. The second one, is that around one every thirty examples are wrong classified given that the detect sensor detects a fail when is a success and otherwise.

7.1.2.2 Changing the object

Little has been said until now about the possibility of changing the object picked by the robot. Firstly, it will be interesting to test if the algorithm gives the same results if the object is different (i.e changing colors, a bigger cylinder, a sphere, irregular forms).

Once it has been proved that the algorithm is capable of providing a good accuracy for other objects, the next step would be to teach the robot to automatically understand that the object to pick has changed.

The first possibility would be not doing anything. The robot may start failing, but as the capacity memory is finite, with every experience the images would be updated and eventually the robot would start predicting well again. However, the epsilon greedy strategy would suppose a problem, as the epsilon greedy strategy would not be reset and the algorithm would predominantly do exploitation rather than exploration.

The second possibility would be to place a button next to the robot, so when the objects to pick are changed, the Neural Network, the replay Memory and the Epsilon Greedy Strategy is reset, and therefore the training starts from zero again.

The third possibility would be to create an automatic reset, to eliminate the manual button and reset all the pertinent modules when the average reward over the last 10 episodes is lower than 60% (for example).

The fourth and last possibility, could be to have several copies of the Neural Network and Replay Memory. Once the weighted rewards on the last episodes is low, the Neural Network stores the current Replay Memory and Neural Network, and starts a training again using a new Replay Memory, a new epsilon greedy strategy, and a new (or not) Neural Network. Therefore, there would be one tuple NN/Replay Memory for every object picked. Once the second NN is no longer making good decisions, the last 10 snapshots are evaluated for all the Neural Networks already built. If the accuracy of a set of weights of a NN is above a threshold, then the object that is now being picked is tagged as "already trained" and thus the old NN is used. Otherwise a new training is launched and a new version of the Neural Network is saved.

7.1.2.3 Testing the algorithm in a physical environment

There is no interest of creating a virtual environment if the algorithm is not eventually tested in a real world. The clearest next step is changing from the virtual environment to the real environment. Even though the results are directionally aligned with the reality, doing these tests would indicate better if a reinforcement learning algorithm is capable of eventually performing better in a production version.

7.1.2.4 Alternating between the Virtual and Physical environment

One of the interests of a virtual environment would be to pretrain the weights of the NN to reduce the time of training in the real environment, thus saving money and resources. Nevertheless, one of the points to be cautious of, is introducing biases to the pretrained weights in the real environment. A difference between both environments may lead to errors in the real environment, as the adaptation may not be perfect.

7.1.2.5 Alternating between a Supervised Learning and a Reinforced Learning

Another interesting project to carry out is to use the existing categorized images of a training to pretrain the weights of the Neural Network, and afterwards use these weights in the reinforcement learning. As a consequence, the convergence of the algorithm would be done offline (not while the robot is working) and could save some time if a dataset is already accessible.

Chapter 8. BIBLIOGRAPHY

Antoniadis, P. (2022, 7 2). *Activation Functions: Sigmoid vs Tanh*. Retrieved from Baeldung.com: https://www.baeldung.com/cs/sigmoid-vs-tanh-functions#:~:text=and%20presents%20a%20similar%20behavior,instead%20of%201%20and%200.

Armesto, L. (n.d.). *Introduction to CoppeliaSim Course | CoppeliaSim (V-REP)*. Retrieved from youtube.com: https://www.youtube.com/watch?v=PwGY8PxQOXY

Arubai, N. (2022, 5 20). *wiki.ros.org*. Retrieved from Documentation: https://wiki.ros.org/

deeplizard. (2019). *Convolutional Neural Networks (CNNs) explained*. Retrieved from Youtube: https://www.youtube.com/watch?v=YRhxdVk_sIs

Gharat, S. (2019, 4 14). *medium.com*. Retrieved from What, Why and Which?? Activation Functions: https://medium.com/@snaily16/what-why-and-which-activation-functions-b2bf748c0441

Lizard, D. (n.d.). *deeplizard.com*. Retrieved from Reinforcement Learning - Developing Intelligent Agents: https://deeplizard.com/learn/playlist/PLZbbT5o_s2xoWNVdDudn51XM8lOuZ_Njv

Nations, U. (n.d.). *Do you know all 17 SDGs?* Retrieved from sdgs.un.org: https://sdgs.un.org/goals

Rastogi, A. (2022, 3 14). *ResNet50*. Retrieved from blog.devgenius.io: https://blog.devgenius.io/resnet50-6b42934db431

Regular API reference. (n.d.). Retrieved from https://www.coppeliarobotics.com/helpFiles/en/apiFunctions.htm: coppeliarobotics.com

reshalfahsi. (2019, 10 23). *arm-suction-sim*. Retrieved from github.com: https://github.com/reshalfahsi/arm-suction-sim

Technology, I. (2021). *What are Convolutional Neural Networks (CNNs)?* Retrieved from youtube.com: https://www.youtube.com/watch?v=QzY57FaENXg

Chapter 9. ANNEX

9.1 SUSTAINABLE DEVELOPMENT GOALS (SDO)

The Sustainable Development Goals provides a shared blueprint for peace and prosperity for people and the planet and into the future. The 17 SDGs are an urgent call for action by all countries – developed and developing – in a global partnership. They recognize that ending poverty and other deprivations must go hand-in-hand with strategies that improve health and education, reduce inequality and super economic growth – all while tackling climate change and working to preserve our oceans and forests.

This project is predominantly affecting one of the SDGs adopted by the United Nations Member States in 2015: the SD

G 8: Decent work and economic work. The SDG 8 is focuses to promote sustainable, inclusive and sustainable economic growth, full and productive employment and decent work for all. This project has some implications covered in this SDG, such as sustainable economic growth and decent work for all.

Many enterprises are still hiring to do monotonous, physical work that could perfectly be done by a robot. This type of tasks is not aligned with founding the life purpose of the enterprise employee's, and thus this robot opens new ways of completing these tasks, and thus allowing the employees to do work more qualified.

Even though this project will use a robot capable of lifting only 5 kg, it is evident that a robot performing a task as monotonous as this one, but for heavy objects, would reduce the physical pain on employees and would definitely drive growth in the life quality.

Additionally, mostly in developed countries, the opportunities of sustained growth due to the increased efficiency when installing robots are numerous. This project, which is not currently subjacent to a single enterprise nor patent, can improve the financial results of all production enterprises, adding thus value to society.